THE UNIVERSITY *of* EDINBURGH

THE FORMALISATION OF DISCOURSE PRODUCTION

by

Anthony Davey

Ph.D University of Edinburgh 1974

ABSTRACT

This paper describes a computer program which produces English discourse. The program is capable of describing in a sequence of English sentences any game of noughts-&-crosses (tic-tac-toe), whether given or actually played with the program. The object is to understand something of what a speaker is doing when he speaks, and the program therefore demonstrates the operation of rules for selecting information into sentences, for connecting sentences into a discourse, and for constructing clauses, groups, and words to convey the required information with the maximum possible economy.

The program uses a systemic functional grammar to co-operate with semantic procedures in producing English. The grammar generates only a limited range of English, but one which is nonetheless sufficient to illustrate the advantages both theoretical and practical of such a grammar for a productive system.

Many other computer programs have accepted more or less natural English input, usually in the form of questions requiring an answer, but few have been designed to produce natural English, particularly connected discourse. As a producing system the present model offers a view of language use from a viewpoint slightly different from that of its predecessors. However comprehension and production are dependent on each other, so that study of one may be expected to illuminate the other.

CONTENTS

Preface

The work reported here was carried out between October 1970 and September 1973 at the Theoretical Psychology Unit of the School of Artificial Intelligence, Edinburgh University. It was financed by grants from the Science Research Council.

It is a pleasure to acknowledge the guidance and encouragement of Stephen Isard and Professor Christopher Longuet-Higgins who jointly supervised the project. The paper in its final form benefeited greatly from Stephen Isard's criticism of earlier drafts.

CHAPTER 1

INTRODUCTION

A speaker must have an understanding of his audience. What he decides to convey, and how he expresses it, depend upon that understanding. In the simplest case, the speaker has information which he believes the audience lacks but would like to have. He gives this information in a monologue. This simple case is examined in the following chapters, with the aid of a computer program modelling the speaker. The program describes, in continuous English prose, any given game of noughts-&-crosses. Our objective is to show how a speaker gets from what needs to be said to the words which say it. The model therefore specifies how to decide what has to be put into words, how to divide this information into sentences, how to arrange a sentence so that its parts fit their context and are easy to understand, and then how to pick words and combine them into phrases to mean the right things. It also specifies, and this is perhaps the most interesting bit, what can be left unsaid: it attempts always to avoid telling the hearer anything he knows already, anything more than he needs to know, or anything he might reasonably be expected to work out for himself. Criteria for this are naturally somewhat arbitrary, but we shall find that the program is generally as tight lipped as possible consistent with saying things that are comprehensible.

First of all, then, the model defines a motive for

discourse. The speaker's knowledge is compared with the knowledge which the hearer is presumed by the speaker to have of the subject matter, and the motive for discourse is to reduce any discrepancy revealed by the comparison. The speaker amends his assessment of the hearer's knowledge as the discourse progresses; in the present model, the speaker assumes that the hearer understands everything he is told, but a more sophisticated model would allow the hearer to ask questions. The discourse ends when the speaker believes that the discrepancy of knowledge has been resolved. So the motive for the discourse is that the speaker should get his model of what his hearer knows to correspond with the relevant parts of his own knowledge.

Having motivated the discourse in this way, we are forced to work out how to define the discrepancy of knowledge, taking account of what the hearer is presumed to have known initially and of what he is expected to infer. We are forced to structure the missing information, and to say how parts are taken from this structure into discourse units. Finally we are forced to say how a sentence and all its constituents are built to do particular jobs; we must say how referring expressions work, and in particular how determiners, modifiers, and pronouns are used. We must attend to the semantics of tense and aspect, and may occasionally use modal verbs.

At the highest level the model embodies a theory of how information within a small universe of discourse is selected and organised into a sequence of English sentences. Because the universe is so small, the task is simplified in three ways. It is easy to define what the speaker must convey to the hearer, namely a sequence of move descriptions; the speaker has only to see how far he has progressed through the game history to find exactly what more there is to say. In a larger context it would probably be much harder to formalise the assessment of the hearer's ignorance; of course, a natural way to assess someone's knowledge is to ask, and to permit him to ask, questions, and any development of the present model should include provision for interaction of speaker and hearer. In the second place, the task is simplified by the fact that the subject matter falls immediately into elements, the moves. Deciding how to structure the missing information just mentioned becomes much more difficult when the universe becomes even slightly richer. The third point is that the relation of each move element to its neighbour is fully defined in terms of the rules and point of the game. A richer context would require formalisation of a larger body of information about desires, expectations, and laws of nature, for example to account for the different conjunctions in:

"I planted roses, but greenfly destroyed them"

"I planted roses, and they flourished"

Charniak (1973) demonstrated how much such factors affect the way we tell a four-sentence story about a children's money-box, and we should expect to find comparable problems for any universe of discourse we might select.

The model incorporates a theory of grammar which is a development from the systemic grammar of Halliday (1961, 1967-8), adapted by Winograd for his remarkable SHRDLU language-understanding system. The grammar is more immediately derived from Hudson (1971) and will be called systemic functional grammar. It is a generative grammar, of a kind which has certain advantages for a language producing system.

The grammar can be thought of as having two parts. The first is an analysis of the grammatical options open to any given item. A major clause item, for example, must be past or present tense, but cannot have gender or number. This analysis is set out in a network of "systems", in which each system is a set of simultaneous exclusive alternatives and the network structure exhibits the logical relation of each system to the rest. So past and present are the two options comprising the tense system, and the network is so constructed that a major clause item must be given a tense from the tense system, and, of course, cannot be given a gender or number. This part of the grammar is explained in Chapter 4.

The second part of the grammar comprises sets of rules which state how the options open to an item may be constrained by the item's role in the grammatical environment. A simple example concerns case terminations. We know that a pronoun such as "we" assumes the accusative form "us" when dominated by a preposition, as in "among us". The grammar therefore contains a rule which constrains a pronoun in such an environment to have the accusative form. The rule is couched in terms of the pronoun's role, or "function", in context. In turn the options selected for a particular item comprise a specification of the grammatical environment of the item's constituents, so further rules derive this environment from the selection made. All of these rules which relate form and function are set out and explained in Chapter 5. They are presented in a formalism which is simple and easy to follow. The computer program in fact interprets these rules of the grammar as commands in a special language, but we shall not concern ourselves at all with the interpreter. The non-specialist will probably find it easier just to think of the rules being deployed as necessary than to follow the working of a computer procedure.

Systemic functional grammar has a certain practical advantage for the constructor of a language processing system. Being generative, it has complete and explicit rules of formation which can be used to govern the

the production of grammatical items. The systems network sets out exactly what grammatical decisions must be taken in order adequately to characterise an item under construction, and the rules in Chapter 5 identify those decisions which are pre-empted by prior decisions about the composition of the grammatical environment. Decisions which are not pre-empted remain to be taken by program procedures which are semantic specialists. For example, the rules tell us that an item which is the object of a preposition must be accusative in form, if the accusative form is distinguished, but they do not say whether the item is to be singular or plural: that depends upon what we are trying to say and in particular upon what referent the item is to denote. The decision is therefore taken by a semantic specialist. The grammar, then, maps the campaign, distinguishing for each constituent those characteristics which are predetermined by the grammatical environment from those which have to be settled by reference to what the constituent must mean.

Another advantage of systemic functional grammar is that it seems to be psychologically more plausible than transformational grammar. Transformational grammarians have normally been cautious in expressing a view about the relation between a theory of grammar and the psychological processes of language use. However, it has been felt worthwhile to search for correlates in psycholinguistic behaviour of certain transformations

(Bever 1971: 435, Kaplan 1971), and the occurrence in many different languages of phenomena which can be given a common analysis within a transformational theory has been given a psycholinguistic significance (Bach & Harms 1968: 113). Within a particular language, idiolectal variation may be accounted for by referring to the varying depths at which a constraint upon a transformation applies, or to variations in the order of rule applications: such an explanation seems to be psycholinguistic as much as formal, (Grinder & Postal, 1971). It is therefore not unfair to mention shortcomings of transformational theory as the basis of a psychologically plausible model, and this we briefly do in section 4 of Chapter 5. Nonetheless, we shall be cautious in preferring the systemic functional grammar, asserting only that it enable the model to tackle problems which a speaker evidently tackles, and not that a speaker has a systemic grammar "in his head".

The present computer program, and presumably a speaker likewise, has information about the job to be done by the next utterance before it decides the form the utterance will assume to do it. The same can be said not only of utterances but also of each smaller constituent of the utterance. But systemic functional grammar rests upon an analysis of the functions performed by each grammatical item in its context, and states the relation between these functions and at least some

of the grammatical characteristics of the item. It therefore corresponds well with the requirements of the productive model, and acquires a certain plausibility in consequence.

Systemic theory, unlike transformational grammar, does not confine itself within the bounds of a single surface - structure sentence. The functions of items are analysed within their context, and the scope of that context is to be as wide as is necessary for an adequate analysis. The grammar used in the program is in fact a very simple one, and so formalises only syntactic functions within the limits of a single surface - structure independent clause. However, the boundaries of systemic analysis are being extended to include the pragmatic and social context of utterance (Halliday in press, Fawcett 1973), and the program's grammar is in principle capable of extension to include these developments. A theory of grammar which accommodates the speaker's need to raise his eyes from the immediate sentence to the surrounding discourse is more plausible than one which doesn't.

We said a moment ago that determination of function precedes determination of form in the model's procedures. The same precedence should probably be true of discourse units larger than the surface sentence. We may recall demonstrations (Sachs 1967) that subjects normally

forget the syntactic form, including sentence boundaries, of heard material very much quicker than they forget the meaning of it. It seems likely that in a similar way the speaker decides the information he wishes to convey, or the social function to be performed, in his next piece of talk before he knows how many sentences he will divide his utterance into. There would in this case be some advantage in a theory of grammar such as the present one.

It is a commonplace that although Molieré's gentleman talked prose, he didn't invariably talk sentences. Completed utterances which are not well formed sentences occur in a variety of circumstances. Others have investigated partial utterances in conversational exchanges but the present model is not interactive and has nothing to say about these things. However, there is one type of ill-formed sentence which, although not produced by the current model, may be illuminated by it. As sentences of this sort are not uncommon in ordinary speech, there is an advantage in having a grammar which might accommodate the model's production of anomalies.

A speaker sometimes fails to foresee that a particular part of his current sentence will be lengthy and complex and so will make the sentence unclear. This is particularly likely to happen in the construction of referring expressions, as the speaker realises the

complexity of the modifiers and qualifiers needed to convey his intended referent. He may remedy the situation in one of a variety of ways, for example by inserting a parenthetical sentence:

"I met Jane's friend who - you know, you met her in Norfolk last year - and she said ..."

Although the present program does not produce utterances like this, but instead produces somewhat elaborate referring expressions, the grammar upon which it is based would accommodate such anomalies in a simple and natural way, because the form of an item is specified as late as possible in the construction process. Whereas transformational grammar prefers to define all the transformations required for a surface sentence before applying any of them, systemic functional grammar determines the structure of each grammatical constituent only as its construction is taken in hand. It is therefore apparent that changes of plan, leading to anomalous utterances like the last example, are likely to cause only a local disturbance, confined to the part of the model responsible for making the constituent concerned. In the case of the last example, the part resonsible for making the referring-expression would break off and make the parenthetical sentence before reporting its task achieved: how it achieved it would not be predetermined or subject to review.

In this way the grammar would accommodate the production of anomalous sentences by delaying a decision

about surface structure until it was inescapable. Planning great extents of surface structure before producing any of it not only implies an unlikely degree of prescience in the speaker, but also makes it harder to explain how anomalies appear at all. Systemic functional grammar, however, seems to provide a framework within which a model might produce anomalies, and for the right reasons. The grammar therefore gains further plausibility, though in this case the reason is not that the grammar enables the model to solve problems which the speaker evidently solves, but rather that it would enable the model, in a natural way, to fail to solve problems which the speaker evidently fails to solve.

To close this introduction, a word must be said about the reasons for casting the model in the form of a computer program. After all, the difficulty of writing a complex program compels the programmer to oversimplify and to take short cuts: he limits the range of choices open to the program, and makes simplifying assumptions. Such criticism is entirely justified, and throughout the following chapters, particularly Chapter 6, oversimplifications and assumptions will be mentioned. Nonetheless, a program may have the virtues of its vices. The programmer oversimplifies because the rules he is specifying for the computer to follow must be explicit, complete, and coherent. Even the very limited and simple grammar incorporated in the present model was improved

from its original state by the computer's demands. The same argument applies, but with greater force, to the semantic specialist procedures. As we have seen, the objective is to show how discourse is constructed to convey information, and we must therefore state not only what grammatical options are open in any particular case, but also how choices are made between them. We must state not only that a noun group may be definite or indefinite, but also how a speaker decides which should be. A verbal account of such a decision, lacking even the formalism of the rules of grammar, would be very liable to error, whereas a computer procedure can be tested not only for consistency but also for adequacy in varying circumstances.

But the most important reason for using a computer and a program to model the brain and mental processes is that a computer seems to be the most brain-like thing we have, and programs the closest analogy to the brain's processes. We need a procedural vocabulary to describe how a speaker gets from his intention to his utterance, and such vocabulary may be supplied from the theory and practice of programming. Winograd (1972) stressed the merits of his "procedural grammar", particularly in connection with what he called demons, procedures responsible for dealing with co-ordinate conjunctions. His program was a team of semantic and syntactic specialist procedures any of which might take charge when called

upon. This concept played an important part in Winograd's suggestions about psycholinguistic processes, and we should note that it derives from advances in programming practice. Another example is provided by Isard & Longuet - Higgins (1973): they made use of facilities provided by the POP-2 programming language to illuminate the relation between clause constituents, and in particular between the verb and nominal participants. Examples from the present program will occur throughout the following chapters, but we might instance here the treatment of certain nominal clause participants. The programming language used allows us to treat procedures as passive objects, rather as the Queen of Hearts tried to use the flamingoes. We can store information in a procedure, or let the procedure store information in itself, accessible to other procedures. We can put a prepared procedure on ice, and at a later time unfreeze it and let it run. This means that nominal participants can be moved into position, symbols in a symbol string, as though they were inert; but then any participant can be called upon to co-operate actively in the construction process. This capacity to be simultaneously a symbolic pigeon-hole and a procedure seems to be worth bearing in mind when we think about the brain's ability to perform computations.

We can, of course, use programming concepts without planning a program, and we can plan a program

without writing or testing it. For example, a generative grammar is a program for operations upon symbol structures but the majority of such grammars have never run as programs in a computer. Nonetheless, programmers know how hard it is to get a program right without testing it, and in particular to foresee correctly the interaction of the constituent procedures of a program such as the present one. The correction and development of a program not only results in a program which works, but may also stimulate new understanding. What the theory owes to the program, then, is likely to be simplicity, clarity, and a procedural language in which to express certain thoughts about psycholinguistic events.

CHAPTER 2

GAME DESCRIPTIONS

Introduction

We come now to examine what the program does. The first three sections of this chapter illustrate respectively the arrangement of move descriptions into coherent discourse, the construction of sentences, and the construction of clause constituents, particularly referring expressions. The various points illustrated are then pulled together in examples of complete game commentaries, and a closing section explains the remarks made by the program when playing a game of noughts-&-crosses.

The examples given in this chapter have all been produced by the program. The program is written in the POP-2 language, and runs under the Multipop operating system on an Elliot 4130 computer. It needs 30K of storage in addition to space required by the Multipop system, and takes between ½ and three minutes to produce each sentence, depending upon the complexity of the calculations required and the length of the final product.

2 Planning discourse

The program gives a commentary on a game, or part-game, of noughts-&-crosses. It assumes that the audience understands the game and follows the commentary as it is given. In order to help the audience, the program arranges the commentary in such a way that each separate

sentence describes a coherent episode in the game, a move or sequence of moves which forms a "play" in the struggle. The program has available certain sequential and contrastive conjunctions with which it signals to the audience the relation of one move to the next, and its deliberations about the arrangement of move-descriptions into sentences are influenced by a preference for making the fullest possible use of these signals. The program's resources include subordinating conjunctions, and so the program may at this stage decide not only what moves the next sentence will describe, but also whether a particular move will be described in a minor clause. We shall see later the circumstances which make this desirable. This part of the program is described in detail in Chapter 6, especially section 3.

The relation between one move and the next may simply be that of valid sequence, which the program conveys by "and":

i) move 1, and move 2.

"You started the game by taking a corner, and I took the opposite one."

ii) move 1, move 2, and move 3 .

"The game began with my taking a corner, you took an adjacent one, and I took the middle of the same edge."

Such a run of moves related only by valid sequence is

not very common in practice, and longer runs, if they occur, are always described in more than one sentence.

Contrastive conjunctions are more varied. A contrastive conjunction warns the hearer that something he had been led to expect didn't turn out that way. The conjunction links the expectation and disappointment into a chunk. A threat foiled is a natural example:

iii) Move 1 but move 2.

"I threatened you by taking the middle of the board but you blocked my line."

There are, however, constraints upon sequences of contrastive conjunctions. It must always be immediately obvious what two pieces of information are being contrasted, and so the following example is confusing:

iv) Move 1 but move 2 but move 3.

"I threatened you by taking the middle of the board but you blocked my line and threatened me but I blocked your edge by taking the middle of it."

The reason seems to be that the first "but" leads the hearer to package the first move and the second move into a chunk, whereupon the second "but" requires him to break that chunk and put moves two and three together instead. The difficulty is really that the second move has both a defensive and an agressive aspect; as Mr J L Stansfield has pointed out in conversation, an attractive

solution is to mention these two aspects in two distinct sentences, as:

iv a) "I threatened you ... but you blocked my line. That threatened me but I blocked your edge."

However, the present version of the system must complete its description of a move within a single sentence because of the way it keeps track of the progress of the commentary. It therefore represents this example by breaking the sentence after the second move. The sentence break makes a heavy pause, after which the third move in a sentence of its own can be contrasted with the whole preceding situation. In sentence - initial position "but" is replaced by "However" as:

v) Move 1 but move 2. However, move 3.
"I threatened you by taking the middle of the board, but you blocked my line and threatened me. However, I blocked your edge by taking the middle of it."

"However" contrasts the information in the sentence it introduces with the situation described by the preceding sentence. We therefore find it confusing if the sentence introduced by "However" includes an internal contrast marked by "but":

vi) Move 1 but move 2. However, move 4 but move 4.
* "I threatened you by taking the middle of

the board, but you blocked my line and threatened me. However, I blocked your edge and threatened you but you blocked my diagonal and threatened me."

"However" directs the hearer to contrast the whole of (move 3, move 4) with (move 1, move 2), whereas "but" contrasts move 4 with move 3. As we have just seen, the system cannot produce:

vi a) "I threated you ... but you blocked my line and threatened me. However, I blocked your edge by taking the middle of it. That threatened you, but you blocked my diagonal and threatened me."

Since the system cannot make the contrasts of the move's two aspects separately, it drops the "However" marking the contrastive link between the two sentences and produces:

vii) Move 1 but move 2. Move 3 but move 4. "I threatened you ... but you blocked my line and threatened me. I blocked your edge and threatened you but you blocked my diagonal and threatened me."

The system chooses to drop "However" on the grounds that sentences are constructed to express the relation of their parts whereas relations between sentences are relatively secondary.

We said that the program signals a contrast only when it is clear what two items are contrasted. The program therefore does not produce:

viii) Move 1, and move 2 but move 3.

* "I took the corner opposite the one I took first, you threatened me by taking the middle of the board but I blocked your diagonal."

This is confusing, perhaps because the hearer is not immediately certain whether the third blocking move contrasts with both the preceding moves, or with just the most recent one. On the other hand, if the contrasted pair of moves comes first in a run of three, no problem arises. The program may produce:

ix) Move 1 but move 2, and move 3.

"You threatened me by taking one of the free corners but I blocked your edge, and you forked me."

The hearer is able to group moves 1 and 2 into a chunk, and then adjoins move 3 as an appendix. In fact such examples are rare, because the factors considered by the program normally dictate other arrangements. The present example is unusual both because it is taken from the description of an unfinished game in which the outcome of the fork is not known, and because the second, blocking move, was purely defensive.

In some cases where two moves within a sentence are contrasted, the speaker wants to warn the hearer not to raise his expectations too high upon hearing the first move of the pair. In such a case the first move may be described within a minor "although" clause, which lets the hearer know in advance that what is about to be said didn't work out. For example, the tactical situation may be such that move 1, though defensive, cannot forestall defeat: the system avoids raising false hopes by producing:

x) Although move 1, move 2.
"Although you blocked one of my edges,
I won by completing the other."

The system takes account of contrast in one other case. It draws attention to mistakes other than those made by itself, and contrasts the erroneous move with the better alternative:

xi) Hypothetical but move 1.
"You could have forked me but you took the square opposite the one you had just taken."

The modal verb warns the hearer, in advance of actually specifying the hypothetical move, that the move didn't really happen, and so not to expect too much of it. It seems equally natural to give the hearer an earlier warning by putting the hypothetical move in an "although" clause instead of in a major clause followed by "but":

xii) "Although you could have forked me, you took the square opposite the one you had just taken."

and the system has the alternative of doing this, but only where the hypothetical move is described in just one simple clause. Where the description of the hypothetical move is more complicated, the system describes it in a major clause incorporating a conditional minor clause, as:

xiii) "If you had completed your diagonal, you would have won..."

and then marks the contrast by "but":

xiv) "...but you took the corner opposite the one I had just taken and so I won by completing my edge."

We notice in passing that where the mistaken move gave the opponent a chance of winning, and he took it, the system inserts "so": this indicates that the win was related to the mistake not only by simple sequence, but actually as a consequence.

Sentence organisation

We have just examined the rather elementary ways in which the system allots subject-matter to sentences, signals where possible the relation of one sentence to the next, and within each sentence give further hints about the relation between the clauses. We now

pursue the same theme in a more detailed examination of sentence organisation.

The sentences made by the system are fairly basic. They comprise subject-verb-complement; the verb may have an adverb with it, and the complement may be null, an adjective phrase, or an object. The structure may be simple or complex; if complex it may be co-ordinate or subordinate. However, the system does not use any of the rhetorical devices which actual users of English constantly employ to guide the hearer, to throw constituents into prominence, and simply to vary the hearer's diet. Some of the things said here might not be true of this wider corpus of English.

Verbs and sentence structure.

In a sentence comprising referring expressions and a verb, the verb works on material supplied by referring expressions. When the verb has operated, the hearer's model is altered. This means that in a co-ordinate complex sentence the order of verbs must be right; the hearer must be able to understand each simple clause as he gets it.

So if an action is described and then enlarged upon, the enlargement generally comes second:

xv) "I took a corner and threatened you."

xvi) "You completed your line and won."

The principle may be overruled to save the hearer from having to re-arrange things which he has already got organised. No square may be taken before the game starts, and:

xvii) *"I took a corner and started the game."

is confusing. Perhaps starting the game is somehow an independent action prior to the move thus re-interpreted, and so

xviii)"I started the game and took a corner."

is preferable.

Sometimes a move has two interpretations of equal interest. In this case the two must appear in the right order. For example, if a move is both parry and riposte, the parry must be disposed of first; the preceding move left the hearer agog over the threat, and to ignore it at the start of this sentence would be to confuse him: so the program produces

xix) "I blocked your line and threatened you."

not

xx) * "I threatened you and blocked your line."

Having made this distinction of the material into basic and re-interpreted actions, the system may relegate the basic one to a subordinate clause:

xxi) "You started the game by taking a corner."

The order of the verbs may violate the principle illustrated in xix) and xx), because the preposition guides the hearer to the correct interpretation. Nothing can happen after the end of the game, yet

xxii) "You won by completing your edge."

is acceptable; the "by" tells the hearer that he is about to be told how the win was achieved, and so about something which preceded it.
If the two verbs are of equal significance, a sub-ordinate structure is not appropriate. The system eschews

xxiii) "I threatened you by blocking your line."

in favour of xix).

Verb selection

The system selects the most significant and most general verb, and omits specific details if it can. For example, in xix) the hearer is left to work out for himself that 'blocking a line' involves taking a particular square, and which square that must have been. Similarly, a 'fork' entails a double or even a treble threat, and so the hearer is not told that the other player was "threatened" by the fork. If there was only one possible way of making the fork, he is left to the same deductions as in xix). An exception to this general principle is that if a move constitutes a

block and nothing more, the system garrulously explains which square was taken to form the block; this exception isn't necessary but seems to improve the system's style.

2.4 Referring expressions

The point of a referring expression is to bring a referent to the hearer's mind. The world is full of things we can refer to, and referring expressions have to pick out the right ones with economy. This section illustrates the simple ways in which the program attempts the task.

2.4.1 Context and implicit antecedents

We rely constantly upon the overall context of our conversations: in a greenhouse a mouse is a predator upon dahlia tubers, not a weight for re-threading sash-cords. So in describing a move the system may say that the player took "the middle of an edge" or the "middle of the board" and leave it to the hearer to understand that in this context the middle of an edge, or of the board, is a square.

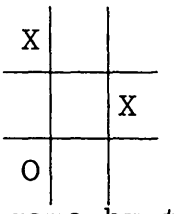
Another economy can be achieved by referring back to entities already mentioned. The mention may have been implicit: for example, if two successive moves have taken adjacent corners of the board, the system identifies the square between them as

xxiv) "...the middle of the same edge"

even if it hasn't mentioned any edge before.

The system not only refers back to implicit antecedents, but also uses explicit antecedents without explicitly referring to them in the anaphoric expression. If a corner has just been mentioned the system may refer to its neighbour as "the adjacent one" without explicitly saying what it's adjacent to: it makes that explicit only when it appears necessary in a context which contains too many distractors.

An earlier version of the system combined both the economy measures just mentioned in a description of the following three moves:



"You began the game by taking a corner, I took an adjacent one, and you took the middle of the opposite edge."

We notice here that what the "opposite edge" is opposite to is implicit; it is the edge common to the two corners mentioned, not either of the two corners themselves.
We notice further that we do not need to make the anaphoric expression explicit either: " the opposite edge" is perfectly comprehensible, and the explicit alternative is actually harder to follow:

* "the middle of the edge opposite that one."

The present version of the program retains the ability to use this form of anaphora, but in practice uses other alternatives in preference.

2.4.2 Pronouns

Pronouns are an economy measure but they are more than that too. Obviously it is useful to be able to refer to "the man who broke the bank at Monte Carlo" as "he" the next time he turns up. But definite pronouns have a deictic component too, and the selection or omission of this is not an arbitrary choice. While chatting to my girl-friend Amaryllis, I refer to her as "you" and the deictic component helps to pick her out. I may not refer to her, though I may address her, as "Amaryllis" because the absence of the deictic component is marked in such a context, and is taken to imply the remoteness of the referent. The absence of a deictic component in the indefinite article can be used in a similar sort of way (Isard & Longuet-Higgins 1970). In

xxv) "John saw a flying saucer. Mary saw a flying saucer too."

we leave the hearer in doubt whether the two saw the same thing: on the whole he would probably suppose that they didn't. If they did, we must use a definite expression with a deictic component to pick up the previous referent, for example:

xxvi) "...Mary saw it too."

The system therefore uses some personal pronouns. It refers to itself as "I" and to the person corresponding with the program as "you". When describing a specified game involving third parties, rather than one which it has just played, it uses "he" and "she" in the appropriate circumstances. It uses "it" when the deictic component is necessary, as:

xxvii) "I blocked your line by taking the middle of it."

and in some cases when a full referring expression would be acceptable, as:

xxviii) "The game began with your taking a corner, and I took the opposite one. It hasn't yet ended."

It uses possessive pronouns:

xxix) "You blocked my diagonal and threatened me, but I blocked yours and forked you."

and the deictic "that" is used to direct attention to the nearer of two confusable antecedents, as:

xxx) "You began the game by taking a corner, I took an adjacent one, and you threatened me by taking the one adjacent to that."

We notice that within complex referring-expressions "it" seems too slight an anaphor in some cases: thus

xxxi) "You blocked my edge, and I took the middle of the one adjacent to it and opposite the corner you took first."

is better if "it" is replaced by "that", as:

xxxi a) "You blocked my edge, and I took the middle of the one adjacent to that and..."

An earlier version of the program formalised this provision by reference to the complexity of the referring expression within which the anaphor occurred. However, the present version manages without this for the moment, although it might need to be reinstated in any development.

The only indefinite pronoun used by the system is "one". The antecedent of "one" is a word-string, as explained in chapter 4 section 5.2, and chapter 6 section 7.6, and so it appears with an article and perhaps a modifier:

xxxii) "You started the game by taking a corner, and I took an adjacent one."

It may disappear altogether in cases where the adjectival modifier is idiomatically capable of standing at the head of the noun-group, as "other" in:

xxxiii) "The game began with my taking a corner, you took the opposite one, and I threatened you by taking another."

Here we notice that the indefinite article is conventionally joined with the following word, in this case "other". A second example of "other" will illustrate two further points:

xxxiv) "The game began with my taking a corner, you took one of the adjacent ones, and I threatened you by taking the other."

The correct treatment of "one of the...ones" is a matter for debate. We regard "one" as a number word, comparable with "two, three,...", in this noun-group, and "ones" as a pronoun; the question is considered in chapter 4 section 5.2. Evidently then, the pronoun "one" takes the plural termination when required. More interesting, however, is the construction "one of the...the other" illustrated in the example. "Other" is an anaphoric adjective, like "same", and its use in the way illustrated binds the major clauses within its scope into a unit to be comprehended as a whole. Its use raises problems for the program, which are explained in chapter 6 section 7.7.2.

2.4.3 Non-pronoun referring expressions

The principle upon which the system is built is that you treat your hearer like the tax-man; give him nothing you don't have to. What you have to give him depends upon the context and the subject-matter. If I say:

xxxv) "I saw a man with two left feet today."

"a man" is a perfectly adequate referring expression, because the odd thing, and what I want to convey, is his peculiar combination of being a man and having two left feet. Whether he was, for example, tattooed or not is irrelevant. It is an extremely difficult problem to formalise this criterion of relevance, depending as it does upon the prior formalisation of the speaker's model of the hearer. It is, however, correspondingly fundamental to the formalisation of language production.

The present system relies upon the fact that the players, and the game, are unique. Of the remaining entities in its universe, the board is also unique. The constituents of the board are mapped by a tactical evaluation to equivalence classes, because the system is trying to describe a game in terms of its tactics. For example, at the beginning of the game, all the corners are the same so far as tactics are concerned, so that at that stage a corner is just "a corner". Later on the system might have to be more specific. If it has to be more specific, it looks for a restrictive addition which rules out all entities denoted by the main noun except the ones which fall into the equivalence class of the referent.

The picture is a little complicated by presuppositions. If a player takes a square, the square must have been free. So if the system is constructing a referring expression to denote a square, the direct object of "take", it produces an expression in which the square is implicitly qualified by "free". For example, if a square has just been mentioned and the system now needs to refer to the only one of its neighbours which is free, it produces:

xxxvi) "...took the adjacent square..."

The referent is unambiguous, despite the fact that more than one square is adjacent. This is the only case within the system where it is necessary to take account of a presupposition inherent in the verb, but the principle is of very wide application and extends, of course, to

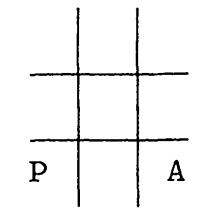
contextual presuppositions established by sentence constituents other than verbs.

2.5 Two examples

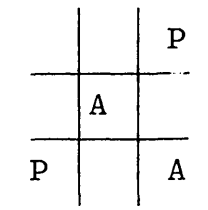
In order to pull together all the strands illustrated in this chapter, we may now consider some complete samples of discourse produced by the program. The commentary is accompanied by a sketchpad illustration of the progress of the game: the speaker's initial is "P" and the human is shown as "A".

The following commentary was given on the moves shown:

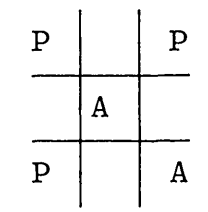
'The game began with my taking a corner, and you took an adjacent one.



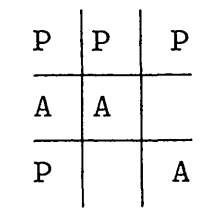
I threatened you by taking the corner adjacent to the one which you had just taken, but you blocked my diagonal and threatened me.



I blocked yours and forked you.



Although you blocked one of my edges and threatened me, I won by completing the other.'

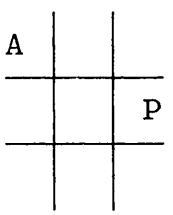


We have already noticed most of the points of interest illustrated in this commentary, but it is perhaps worth drawing attention to the economy of the discourse. The description applies to the game illustrated and to any game which is tactically equivalent. Furthermore, the commentary requires the audience to follow intelligently: we observe, for example, that after the third move, the

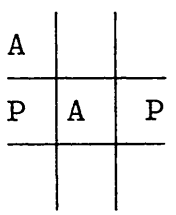
the program never says that any particular square was "taken", but instead uses terms like "block" or "complete" which are unambiguous provided the audience follows the commentary. We might finally note that the program has mentioned a threat presented by the penultimate move, despite the fact that it was a vacuous threat in view of the tactical situation. Another version of the program does not mention such threats, and it is debatable which produces the more natural English; perhaps they should be mentioned as threats only if the opponent's win does not immediately follow, either through his omission or because the game is unfinished.

A second example illustrates the problems of using pronouns.

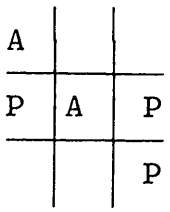
"I started the game by taking the middle of an edge, and you took an end of the opposite one.



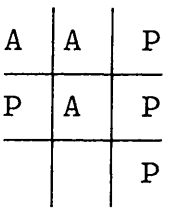
I threatened you by taking the square opposite the one I had just taken, but you blocked my line and threatened me.



However, I blocked your diagonal and threatened you.



If you had blocked my edge, you would have forked me, but you took the middle of the one opposite the corner I had just taken and adjacent to mine and so I won by completing my edge.'



The program here has used "mine" to mean "my edge" but a moment later denotes the same referent by "my edge" in "... completing my edge". It did this for reasons set out in Chapter 6 section 7.6: in summary, "mine" cannot be repeated in

* "... completing mine"

because potential antecedents for the anaphor are too far away either 'horizontally' along the word string, or 'vertically' in constituent structure. The pronoun specialist is sensitive to slight alterations in the rules, and the results illustrated here represent a cautious compromise between wordiness and ambiguity of reference.

If the program is required to play a game before describing it, certain standard remarks may be produced at the start and finish. The motivation of these

remarks is primarily practical: we want to know who is to move first, and when the game is over. However, the opportunity was taken to illustrate the use of modal "will";

"Will you start the game?"

imminent aspect:

"I am going to start the game"

and perfective aspect when the game ended but is not yet done with:

"I (you) have won the game"

In the subsequent game commentary, the same information will be conveyed by the simple past:

"I (you) won the game"

Finally, the passive is used in the case of a draw:

"The game has been drawn."

These standard comments have no intrinsic interest, but they offered a sensible method of extending the range of clause options which the system might exemplify.

Further examples of game descriptions and of output while playing a game with the program are presented in Appendix 1.

CHAPTER 3

A REVIEW OF PREVIOUS SYSTEMS

Introduction

Since 1960 there have been upwards of twenty projects which have used English of varying degrees of naturalness for communication with a program. Unlike the present system, very few of them were meant to model a speaker, although several were incidentally capable of producing English sentences cobbled together in simple ways. So this review will first consider how well each system models an understanding hearer; to define a hearer is to define at least part of the speaker's task - he must say things which the defined hearer can be expected to understand. The second section of the review examines the contribution to linguistic theory made by some systems which have formalised the phenomena of "local context"; by this we shall mean approximately the scope of anaphora. The third section is concerned with syntax.

2. The understanding hearer.

In this section we will scrutinise the relation of each system's information store to the understanding process, attending in particular to two questions.

The first question is whether the system guided the comprehension of fresh material by using facts it knew or had picked up. Most understanding systems have had a model of the universe of discourse in terms of which input is interpreted and output may be produced. A

majority of the systems reviewed here, however, kept the syntactic analysis of input rather strictly apart from the retrieval of information; the first stage used dictionary lookup and a parsing algorithm, and then the retrieval of information was done by separate procedures, using the data handed over by the first stage. Such separation of analysis from comprehension cannot be successful with complex input; syntactic ambiguities must be resolved as soon as possible by an appeal to what the input might mean.

The present system says things which the hearer can understand only if he has understood and remembers what he has been told. For example, when it mentions in succession:

> "A corner...an adjacent one...the middle of the same edge."

it assumes that its hearer is understanding as he goes through the sentence and so can identify the "edge" referred to. More generally, when it says:

> "I won by completing my diagonal."

it presumes that the hearer has understood to game so far, and so knows which "my" diagonal must be.

The second question we shall ask is whether the user could tell the system anything. Some systems accepted declarative input as well as questions. Declaratives gave these systems generic truths or definitions of terms for

addition to the store. They were almost always in a fixed format analysable without reference to their meaning and in some cases could be given to the system only when it was in a special mode. We shall prefer as models of the hearer those question-answering systems which accepted mingled questions and declaratives.

2.1 Fixed-format systems.

Following Winograd (1972:34) we classify as fixed-format five systems which selected from input just the information which fitted their predetermined schemata. We shall pass over SAD-SAM (Lindsay 1963a, 1963b), BASEBALL (B F Green 1963), and SIR (Raphael 1968) very briefly. None of these systems was intended to model a language user. BASEBALL could not be told anything, and analysed input by purely formal means, and SIR was made to investigate:

> "the ability of the computer to store and utilise relational information in order to produce intelligent behaviour." (p.58)

Raphael further said that the handling of input English was:

> "independent of the representation and retrieval problem"

where his primary interest lay. Raphael is actually unfair to himself, as SIR was sometimes able to disambiguate 'has as a property' from 'has as part' by referring to relevant information acquired from the conversation,

and a speaker could rely on that in composing things to say. However, Raphael was clearly not trying to produce a model of a hearer in a conversation; he was making a Semantic Information Retrieval system which as a concession to the user accepted something like natural language as its input.

SAD-SAM picked up family relationships which were referred to, or explicitly stated, in input, and added the information thus gained to a stored family tree. The system could print trees which resulted from accumulation and inference and thus exhibited a type of McCarthy's (1958) "common sense", but, as in SIR and BASEBALL, syntactic and semantic analysis were quite separate. It is symptomatic of Lindsay's intentions that he draws no attention to the rather nice ability of his system to learn from reference and presupposition as language users actually do; he is much more concerned to explain the working of the inferential component.

STUDENT (Bobrow 1968) was, its author claimed:

> "...the first computer realisation of a theory of discourse analysis...that maps a discourse onto some representation of its meaning."

and not just a clever data-management system. STUDENT took problems in elementtary calculus stated in English discourse, converted them to a representation in simultaneous equations, solved the equations, and printed the

results. If the set of equations was insoluble, STUDENT tried to substitute for possible idioms from a store of equivalents, and if that failed, it asked for further equations to be stated between specified terms. STUDENT justifies Bobrow's claim for it; we shall return later to its use of pronouns, and here confine ourselves to two comments. First, STUDENT didn't have a store of information in quite the same way as most other question-answering systems. Apart from its knowledge of idioms and units of measurement, every fact it knew was given in the set of equations derived to represent a particular problem. Solving the set of equations corresponded to the integration and use of its knowledge, and thus to STUDENT's "common-sense". Secondly, the permanent base of idioms could be extended only in a special REMEMBER mode, so that if STUDENT asked for additional information and was told, for example, that:

> "(THE WEIGHT OF A SHIP'S CARGO IS THE DIFFERENCE BETWEEN THE GROSS WEIGHT AND THE NET WEIGHT)"

this information wasn't REMEMBERed for next time.

ELIZA (Weizenbaum 1966, 1967) modelled a psycho-analyst questioning a patient. It relied upon an effective analysis of the focus of questions in certain formats, of the way certain conversational tropes might be interpreted, and of the guidance afforded to the psycho-analyst by the appearance of keywords in the patient's remarks. It therefore attempted to formalise certain aspects of

conversational strategy, with considerable success, but it understood nothing of the input, and relied so much upon holding the conversational initiative by means of questions that one feels impelled to cry with Dr. Johnson:

> "Sir, the art of conversation does not consist in unmeaning interrogatories."

2.2 Text based systems.

This generation of systems is represented by PROTOSYNTHEX I (Simmons et al. 1966), Quillian's Semantic Memory (1968) and the anachronistic story understander of Tharp (1969). All of these systems stored text, more or less directly derived from declarative input, and used the store of text as their model of the world. Questions were answered by looking up phrases from the question in the store. Although Tharp used some interesting heuristics to precis input, discarding less central input, none of these systems understood their input to the extent of being able to reason from it. For example, Lindsay's (1963) system knew that if F was S's father, then S was F's son, but none of the text-based systems had the machinery for inference of this type, upon which a speaker constantly relies in his hearer. Simmons and Quillian went on to develop versions of their systems which to some extent digested the text, having concluded that understanding and some deductive power are essential attributes of a system for storing and retrieving information.

2.3 Limited deductive systems

The next generation of systems, falling between 1966 and 1970, tried to combine the generality of text-based information storage with deductive power such as was possessed by the fixed-format systems of Lindsay, Green, and Raphael. The user was to define relations appropriate to his subject, employ these in storing the information, and then use the relations between stored bits of text to make inferences. What this meant in practice was that the system permitted the user to input an arbitrary set of definitions, and perhaps a body of facts, by way of a set of simple declarative sentences. These sentences had to fit within a fairly narrow range of formats, and were interpreted as phrase-relation-phrase. The system then translated this input to a normalised form in graph (Thompson's DEACON 1966, Kellogg's CONVERSE 1968, Shapiro's SAMENLAQ 1969, Quillian's TLC 1969, Simmons' PROTOSYNTHEX III 1970) or tabular (Colby 1969, Simmons' PROTOSYNTHEX II 1966) storage.

PROTOSYNTHEX III accepted the quantifiers "all" and "some" in input, but didn't really capture their meanings. The universal quantifier could apparently be used only in generic statements:

"All voles like gladioli."

and not where it had extensional force:

"All those men you met last week...".

The use of "a" as indefinite:

"I saw a vole this morning."

was the only use recognised, and the system did not capture its generic use:

"A vole is the gardener's deadliest foe."

The system had no means of capturing the scope and order of quantifiers, and so could not distinguish:

"All voles acknowledge some crimes."

from:

"Some crimes are acknowledged by all voles."

It could not distinguish count from mass nouns, and treated "some" as having the same force in:

"John went off with some beer of mine."

and in:

"John went off with some friend of mine."

The representation of quantified expressions is an extremely difficult problem; the present system does not employ quantifiers and we will leave the matter there.

CONVERSE, SAMENLAQ, and PROTOSYNTHEX III extended their systems to permit relations to appear as terms in relations. This meant that they were able to represent embedded sentences, such as the clause forming the subject in:

"To give a good thing is to reward."

(PROTOSYNTHEX III)

But in none of these three systems was the criterion for selecting a concept to be a relation, or the reason for deciding that all relations were two-place, made clear, so we are left in the dark about what this extension really achieved. More generally, all the systems of this generation suffer from an inadequate theoretical foundation; we have no characterisation of the kind of inferences which their storage systems permitted, nor any proofs of the completeness, consistency, or otherwise of the deductive component.

DEACON and CONVERSE were capable of guiding syntactic analysis by accessing the information store, although CONVERSE permitted access only to definitions and general truths, not to specific facts. In this respect it resembled PROTOSYNTHEX III, which checked each potential triple of term-relation-term against a paradigm or "semantic event form"; a set of paradigms had to be input in advance before the system could look at any English. So PROTOSYNTHEX III didn't guide understanding by reference to what it had been told, despite its stated objective of integrating the semantic interpretation and deductive retrieval components.

CONVERSE was the most impressive of this generation of systems. It worked by compiling English questions into programs of arbitrary complexity for looking at stored information, so although the store was fairly

simple the system could answer multiple questions such as:

> "What is the highest and lowest median family income for the eastern cities and the western states?"

which it understood as a request for four numbers. The question is, of course, ambiguous, and, with less obviously incompatible superlative adjectives, multiply so. For example:

> "What is the largest and fastest vessel in the Swiss and Hungarian navies?"

CONVERSE was in principle capable of calculations and output of arbitrary complexity, but in practice seems to have lacked a sophisticated tool, such as PLANNER or an augmented transition network formalism, to analyse English. It therefore ignored the subtleties of the input, a sergeant-major chairing a theological congress.

Quillian's (1968) semantic network formed the basis of the Teachable Language Comprehender (1969). The user primed TLC with a coded network, and then presented it with a noun-phrase or a very simple sentence; the system responded with an explication of the input in terms of the information stored in the net. The system had no deductive power, and remained fundamentally a storage system for partially processed text rather than meanings, similar to

Quillian (1968). We may accept the goal of Quillian's research, namely the understanding of references in terms of existing knowledge, as part of what an understanding hearer does, without liking his method of reaching the goal.

Several systems which used natural language input can be left out of account here. QUAC (Ambler and Burstall 1969) and QUAD (Breslaw 1969) were both relatively small scale investigations of techniques of computation rather than of language and understanding. SAMENLAQ II (Shapiro 1969) was, like SIR, really a system for storing and retrieving information; unlike SIR, it allowed the user to define his own relations but it was not a linguistic theory, and its range of permitted inputs, in the form of two-place relations, looks less like English the more one examines it. Finally Colby's artificial belief system (1969) was essentially a text-indexing system of the previous generation, with a limited deductive capacity relying principally upon the relation of set-inclusion.

The weakness of this generation of systems was that they didn't understand their input; with the exception of CONVERSE, the most they could do was to answer questions with the words they had stored, inter-relating them more or less directly by the defined relations. These shortcomings have now been recognised, and Simmons has

turned from the SYNTHEX line of research to consider the problems of specialised semantic representation as a pre-requisite for further advance. Coles (1969), arguing from his experience with the ENGROB system discussed later, urged that more attention be given to the structure of our perception of the world and to its relation to language.

2.4 General Deductive Systems

The problems of the systems just reviewed have aroused interest in finding a representation of meaning which meets at least three requirements. It must actively assist in the understanding of input; instead of being a bank of unresponsive pigeon-holes it must indicate its desires as actively as a four year old to Father Christmas. For example, Schank's (1970) recent conceptual analysis system refers input to a conceptual paradigm, which has been partially matched by the input so far, to see what's missing and so what more to expect. Secondly, if two inputs have related meanings, the storage system must relate the stored representations in a useful way. In practice this means that the input must be understood in terms of an existing semantics, and it rules out the simple tabulation in storage of unprocessed, or partially processed, text, because relations which are explicit in the input are not the only ones which will be needed in making inferences and answering questions. Thirdly, and closely following

the last point, the system must make it easy to reason with the information stored. This is a very tough problem, and one which we shall not consider here, since the present system neither performs nor presumes in the hearer this kind of problem-solving.

But having mentioned the difficulty of characterising the adequacy, consistency, and completeness of limited logic systems, we should add a note on the more recent systems which have performed deductions based upon resolution theorem proving. These have normally preserved consistency and completeness by careful choice of heuristics to govern the choice of axioms, but we should consider whether such systems are in practice or in principle satisfactory models of human cognitive procedures. Winograd (1972) has harsh words to say about the cataleptic inefficiency of proofs deduced from more than a very small set of axioms. Coles (1972), having worked with the question - answering systems QA3 and QA3.5, makes the further point that these techniques are best suited to lengthy chains of inference from a small set of axioms, whereas in problems of understanding language or retrieving information we want techniques which are good for making rather shallow inferences from the relevant members of a very large set of facts.

Against this background we may project R2 (Biss, Chien, & Stahl 1970), SHRDLU (Winograd 1972), GRANIS (Coles 1967) and ENG- (Coles 1969 - 1972). About R2 there is little to say; it is a "cognitive information retrieval system" which is designed to translate input into a "new high-order calculus", but the means it uses are purely formal and seem unlikely to work satisfactorily on any substantial scale.

Coles (GRANIS, ENGDRG 1969, ENGROB 1969, ENGLAW 1972) has always exhibited a keen awareness of the complexity of real English. GRANIS accepted line diagrams and verified the truth of English statements about them. Both graphical and linguistic input was converted to a common predicate calculus representation within which all reasoning was carried out. In this system translation from English into the common notation was interleaved with formal syntactic analysis, and ambiguities of analysis which survived this were finally resolved against the graphical datum in a sensible order: first preference was given to interpretations which took sentence constituents with their closer neighbours, so that in

> "Each resistor in parallel with a capacitor which is ten ohms..."

"capacitor" would be assumed to be "ten ohms" at first, and only when this was marked electrically anomalous

would the resistance be assigned to the more distant resistor. In turn, a semantically acceptable, and true, interpretation would be rejected on pragmatic grounds if it were vacuous and an alternative were available. Coles doesn't give an example, but he has in mind cases such as:

"Each component in parallel to a resistor which is a resistor..."

where it is vacuous to take "resistor which is a resistor" as a referring expression. This is clearly a sophisticated system, embodying some sound observations about the intimate relation between syntax and semantics. In 1968-9 he produced the first of the three ENG- series systems, and in 1972 was working on the most powerful, ENGLAW. Despite the refinement of the parsing system, the power of the QA3.5 theorem proving system, and the depth of analysis which lies behind the canonical set of predicates, functions, and constants, the ENGLAW system is not different in principle from GRANIS. Indeed, despite a mighty expansion of the lexicon in ENGLAW, GRANIS seems to have been able to digest more complex syntactic structures than can ENGLAW.

The third "general deductive" system is the SHRDLU system of Winograd. This system, unlike R2, ENGLAW, and the great majority of systems under review here, is a theory of language, not a data-retrieval or deduction-making machine. Careful attention to the way utterances

are organised to carry information had a determining influence upon the architecture of the input analysis component, and was a precondition of Winograd's achievement. For the moment we are considering only the use of "common-sense" in understanding and replying. The term really means "specialist knowledge", and a central feature of Winograd's system is that syntactic procedures had access to semantic specialists, cobblers not ashamed to stick to their lasts. So SHRDLU was good at guiding understanding by referring to its model of the world, and it digested in an appropriate way the very simple forms of new information which FRIEND could supply. Coles (1972) comments that the semantic specialisation coded into the system has in practice made it hard to transfer Winograd's system to a different context, but a lot of the success of SHRDLU compared with its predecessors is due to the fact that it knows what it is conversing about, and it is not surprising that the formalisation of other, more complex, subject matter is difficult.

2.5 Other projects

Two further projects not falling within this classification scheme remain. LSNLIS (Woods et al. 1972) originates in Woods' (1968) projected question-answering system. The LSNLIS project is comparable in power with ENGLAW but contributes surprisingly little to our understanding of language. The user can't tell it anything in

English, and the analysis of input can't get at the information store, which is actually too large to fit in the computer at the same time. Instead the input is first assigned a syntactic structure; as each S and NP node is completed, an interpreter cashes its subtree into members of a canonical set of functions and their arguments. Some things get done wrong. Given:

> "What is the average analysis of olivine for iron?"

the system first assigns "for iron" to "olivine", the nearest NP, in syntactic structure, and then relies upon the interpretation of "analysis" to pluck "for iron" from its distant subtree, and so to code the canonical "analysis for iron". This is not a satisfactory procedure, and it would probably collapse if faced with input as syntactically complex as that understood by Winograd's SHRDLU. The semantic guidance available corresponds in principle to that of PROTOSYNTHEX III's "semantic event forms", and we may feel that this is inadequate.

Bruce's CHRONOS (1972), at the stage at which he reported it, was primarily a formalisation of the semantics of tense: Bruce himself points out weaknesses in his treatment of aspect, modality, and repeated or frequent events, and we shall ignore these here. The remainder of CHRONOS is comparable with the Party, or

Waiting-for-Cuthbert pilot program of Longuet-Higgins (1972). There is in fact a close correspondence between the definitions of tenses in terms of the relations "before" and "after" of Bruce, and the algorithms used by Longuet-Higgins within his system. However, whereas CHRONOS accepted simple tensed declaratives and noted their contents on a time line, the Party program was supplied in advance with information about the order of guests' arrival at a party. Both systems subsequently let the questioner set "NOW" at any point on the time line, and responded properly to questions requiring the comprehension of present, past, and past-in-past (or pluperfect) tenses, the imminent and perfective aspects, the modal future, and the calibration of time references by the use of time clauses and prepositions. CHRONOS understood about periods of time though not, as we have seen, about repeated actions within a period. Both systems dealt more fully with the semantics of tense and aspect than SHRDLU, the only other system to attack the problems, though neither system understood anything else of the input. Subsequent work by Longuet-Higgins and Isard (Isard & Longuet-Higgins 1973, Longuet-Higgins 1973 mimeo, and Isard forthcoming) has extended the analysis of tense and aspect to modal and hypothetical expressions, and has influenced the design of the present system in these respects.

The Party program was a truly conversational program, and we shall return to it in the section dealing with

"local context". We conclude the present section by observing that SHRDLU and the Party Program alone have formalised some solutions to the problems associated with presuppositions, and their possible falsity, making intelligent use of their models of the world to do so. SHRDLU objects to words which have not been defined to it, to references to events which didn't happen, and rebuts false assertions in detail. The Party program actually corrects misapprehensions, betrayed, for example, by a past time reference to a future event. Presuppositions, like drains, have a pervasive if normally unnoticed influence upon their environment, and any future program which is to model conversational behaviour will have to pay close attention to the subject.

3. Local context

"Local context" will be taken to mean the scope of such phenomena as anaphora. We shall see how a few systems have captured the capacity of an utterance to rely upon the context established by itself or by the words immediately preceding.

3.1 Sub-contexts

Under this heading we shall consider Weizenbaum's revised ELIZA (1967), and Schank's Conceptual Parser (1971). ELIZA used conversational subroutines; for example, within the overall context of Rogerian psycho-analysis, a

keyword such as "hurt" in the input would establish the pain "sub-context" and so set in train the appropriate questions. The perception of a new keyword might switch the system into a fresh "sub-context". Of course, the notion isn't rigorously defined, so that the levels of "context" and "sub-context" remain arbitrary. However, the idea of a conversational sub-routine is attractive, and in a recent system of Power (1974) it has re-appeared; with apologies to Wittgenstein he calls it a "game", and finds it a powerful tool for the analysis of the structure of a conversation.

Schank (1971) has mapped out but not fully programmed a "conceptual parser" which would select the right conceptual context within which to understand further input. His proposed system would attempt to tune itself to the speaker's wavelength from moment to moment so that, for example, the context of cooking or of a Glaswegian brawl would determine the expectations to be associated with "knife".

3.2 Anaphora

A few of the systems under review accepted pronouns in input, and one or two accepted anaphoric adjectives such as "this (result)" and "other" as in:

"Cetti's warbler, unlike any other warbler..."

However, anaphoric references can be understood by purely

formal or syntactic means only in the simplest cases, and so every system except Winograd's SHRDLU (1972) has handled anaphoric reference, if at all, by devices which greatly simplify the real problems. No system which stores the words of referring expressions rather than meanings can tackle these problems.

Bobrow's STUDENT (1968) achieved its impressive results by picking up the most recent variable not yet incorporated into an equation as the antecedent of any phrase introduced by "this", and it got the antecedent of a pronoun by a string-matching procedure. PROTOSYNTHEX (Simmons, 1970) accepted anaphoric adjectives such as "other", but treated them exactly as one-place modifiers such as "red" or "old". Tharp's (1969) system accepted definite pronouns, but apparently only where formal criteria such as gender rendered the antecedent unambiguous. The R2 (Biss et al., 1970) system designers, without explicitly mentioning this problem, appear to have intended to handle not only pronouns but also such anaphoric expressions as the "the officer" in:

> "If a policeman stops a motorist, the officer may demand his licence."

but they offer no indication of how this is to be done within a system that essentially stores text. LSNLIS (Woods et al., 1972) understood the anaphora of "those"

in:

"How many samples contain olivine?"

"Give me those samples."

by searching the previous input for a match for "samples". Finally Longuet-Higgins' Party (1972) system understood "he" in a question as denoting the partygoer most recently mentioned by the questioner, although, as the author points out, in the search for an antecedent the system ignores what it has said itself.

Winograd's formalisation of the way we use local context to help us understand utterances is a great advance over any previous system, and any criticisms here represent, to adapt Huxley, merely our normal reluctance to congratulate the inventor of the motor-car, preferring as we do to carp at the stiffness of the accelerator pedal. His system formalises many aspects of the use of pronouns, and can deal with partial utterances.

3.2.1 Pronouns in Winograd's system

Each pronoun type has as associated procedure which is a specialist at finding the pronoun's antecedent. The specialist can examine the syntactic structure of the current sentence, and references made in previous sentences to times, events, places, and objects. Winograd

notes that we prefer not to use the same pronoun with different antecedents within a single sentence, and that where pronouns have antecedents at a higher or a lower level of embedded clause, the non-reflexive form replaces the reflexive in certain circumstances. He captures one reason for selecting "that" in preference to "it" or "them" in conversational exchange, namely that the antecedent of "it" may be sought in the last utterance of the current speaker, whereas "that" picks up the most recent suitable antecedent. Thus:

Friend: "Why did you pick up the red pyramid!"

SHRDLU: "To clear off the green cube."

Friend: "Why did you do that?"

However, he is too restrictive about "it" and "them". These pronouns are unmarked for proximity of antecedent, and so in the exchange:

Friend: "How many blocks* are not in the box?"

SHRDLU: "Four of them."

Friend: "Is at least one of them* blue?"

the antecedent of "them*" might be the referent of "blocks*" as Winograd (p.10) has it, or it might be the four blocks not in the box. "That" is marked for proximity and so:

Friend: "Is at least one of those blue?"

is unambiguous. The present system uses "that" to select the nearer of two possible antecedents, otherwise "it", according to the slightly weaker rule given here.

Perhaps the most interesting analysis is that of "contrast" in the use of the substitute pronoun "one", as in "... the large one". "One" requires not an antecedent referent, but an antecedent word-string, and the search for the word-string may be guided by a contrast between a modifier in the current phrase and a modifier in the antecedent. Thus in "the large red block ... the little one" the second phrase represents "... the little red block", not "... the little block" or " ... the little large red block", since "large" contrasts with "little". The present system uses "one" in a way which makes clear that the contrast may be marked not only between modifiers drawn from a list of pairs of opposites, as in Winograd's system, but also between the presence and the absence of a modifier:

"a corner ... the opposite one."

There are of course more complex issues here, concerned with the order in which adjectives occur before a noun, and the consequent effect upon the permissible composition of the substring selected by "one". For example, in

Friend: "Pick up the large red block."

SHRDLU: "OK"

Friend: "Now pick up a blue one."

" ... a blue one" represents "... a blue block" not "... a blue large block" or " ... a large blue block".

But we shall leave these more detailed issues here.

3.3 Partial utterance

The only systems to formalise the comprehension of partial utterance are the Party program of Longuet-Higgins (1972, Isard & Longuet-Higgins 1973), and the SHRDLU program of Winograd (1972). Our system produces none. The Party program attended to the tense, aspect, subject, and time-reference of a question, and was able to complete partial questions from the preceding context where permissible, thus achieving:

"Has Bill come yet?"

"No."

"Has Al?"

"Yes."

SHRDLU maintained a much more complex local context, and we have already noticed some of the ways this context could be used to explicate a current reference. Its capacity for understanding partial utterances is not clearly defined, but extended to coping with:

Friend: "When did you pick up the little pyramid?"

SHRDLU: "While stacking up the red cube and a large green block."

Friend: "Why?"

SHRDLU: "To clear off the green block."

Notice however that SHRDLU interprets "Why?" as "Why did

you pick up ..?" which is not the only thing it could represent. Where SHRDLU's response to the first question is lengthy, we may forget the form of the question, and so be less easily guided by the resemblance of "When did you ..?" and "Why (did you...)?". In such cases SHRDLU's most natural response to the "Why?" might be:

SHRDLU: "Why what?"

Finally we may note that Winograd's soundest observations often appear in the partial responses of his system:

Friend: "Is the box supported?"

SHRDLU: "Yes, by the table."

which depends upon his analysis of the "focus" of a question.

4. Syntax

The present system is a system which gets from something needing expression to the words which mean it. This is described more fully in another chapter. Here we mention several factors which drastically curtail our interest in the syntactic components of most earlier systems. First, all previous systems have been designed primarily to understand, whereas this one is meant to produce. A theory of grammar, mapping between meaning and expression, may in principle be direction-independent as transformational grammarians would normally have it, but the practice is very different (Winograd 1972 p.42). Second, ours is

not a sentence-based utterance system. Of course, it produces sentences, but at the deepest level the system is discourse-based, and in this it differs from all current theories of transformational grammar whose deepest level is the representation of the sentence and which have nothing to say about where that came from. Third, having selected the semantic content for the sentence, the system sets in train a meaning-driven production process at the start of which the form of the future sentence is not fixed and may indeed be unpredictable. Such a procedure is obviously not expressible in a deep structure comprehensively marked for obligatory transformations. Fourth, Chomsky (1972 p.15), while attending closely to the meaning of utterances in order to specify the form of transformational rules is determined (p.19) not to permit the rules to have semantic content. The present system, and more generally systemic grammar, is very anxious to account for the form of sentences in terms of what they are for, namely meaning things. For example, the sentence:

"Bill kicked the ball."

is about Bill, while the sentence:

"The ball was kicked by Bill."

is about the ball. Systemic grammar formalises this observation, transformational grammar has no comment. Even if we take seriously the schisms among transformational grammarians, and Katz (1971) is inclined not to, it is clear that a language for programming formal operations

upon tree structures is not going to help our system in its primary task.

For these reasons we shall here say nothing about those systems which have used simple pattern-matching, context free grammar, or some kind of transformation-unpicking procedure for understanding input. These systems are surveyed in Bobrow (1967) and Winograd (1972). This section is ended with some comments on the systemic grammar embodied in SHRDLU (Winograd 1972), and on the augmented transition network formalism of the LSNLIS system (Woods et al., 1972).

4.1 Systemic grammar in SHRDLU

Winograd asserts that his system is based upon systemic grammar (1972 p. 16) and we shall now try to say what systemic grammar does for SHRDLU. As the use which our system makes of this theory is described in Chapters 4 and 5 remarks here will be very brief.

Whereas transformational grammar is preoccupied with constituent structure, and tells how to map between deep and surface structures in steps which are mathematically attractive, systemic grammar is scarcely concerned with that kind of structure at all, and does not have an abstraction which could be called deep structure (Hudson 1971 p.15). SHRDLU's grammar contains just three levels of "rank", namely clause, group, and word. Its parsing

in terms of units at given ranks is therefore a flattish and relatively uninformative tree. Of course, units may be "rank-shifted", so that for example a clause may realise a noun-group; this makes the tree more complex but still not our main interest. The strength of the grammar lies in the 'systems' of choices among features which units at each rank may have; these tell SHRDLU what the present units could be doing in the sentence and how to find out more. To take a very simple example, if the unit is a CLAUSE the system of features is so arranged that only if the clause is MAJOR does SHRDLU look to see whether it is IMPERATIVE, QUESTION, or DECLARATIVE: these features correspond to the illocutionary act assumed to be performed by the current utterance, and so never apply to MINOR embedded clauses. A fuller analysis would distinguish between a description of the illocutionary act and a description of the clause form, whether MAJOR or MINOR.

In analysing a unit there may be several independent systems to traverse: for example, given a noun-group we want to know whether it is pronoun or a noun, and simultaneously we have to know whether it is the subject or part of the complement of the clause.

As a matter of fact, SHRDLU does not proceed straightforwardly through each system in a "top-down" fashion. The parsing procedures constantly appeal to semantics,

and this may result in interruptions, jumps, and recursion in the procedures. Nonetheless, what SHRDLU owes to systemic grammar is the systematisation of the options available within each unit of an English utterance. Winograd built only a simplified version of Halliday's (1967) now superseded grammar into SHRDLU, but even this gave his system a great advantage over its predecessors.

4.2 Augmented Transition Networks

The LSNLIS (Woods et al., 1972) system does its parsing with a recursive augmented transition network (RATN). Woods (1969) claims advantages for this formalism, while Winograd (1972) has reservations. We shall examine the problem very briefly, paying attention in particular to the question whether it is a good formalism for expressing a theory about language.

A RATN is a finite-state transition network, but with three additions which greatly increase its flexibility and practical power. First, arbitrary conditions may be set upon arcs to permit or block the arc-transition. Second, registers may be filled and examined upon making a transition. Third, control may leap to arbitrary points within the current net, or to the start of an independent net, whence it may return to its branch point.

Woods (1969) claims perspicuity, efficiency, and psycholinguistic plausibility for the representation.

It is perspicuous because it the network is easy to understand. It is efficient because it allows re-write rules whose right hand sides have sub-strings in common to be represented as sharing a common sub-net. If the sub-strings are similar, but not identical, noting essential information in a register may yet enable the same economy. Perhaps the most interesting claim concerns the plausibility of the RATN representation for capturing our intuitions about the understanding process. The programmer can often arrange to delay a decision about a basicfeature of the input until the requisite information is to hand. For example, we don't want to have to guess whether the sentence is active or passive before the verb has been reached. Secondly, the RATN has registers which can hold bits of parsed structure. This means that the system can make good use even of a wrong parsing, perhaps needing simply to shuffle the contents of registers.

However, the RATN has so far been used only to implement transformational grammars, essentially unpicking surface structure into a deep structure representation. It is good for this purpose, because the syntactic structures can be presented clearly in network form. But SHRDLU and the present system use systemic grammar, in which, as we have seen, the constituent structure by itself is not very informative. The interesting information is contained in the path selected through the systems of options, and these are not naturally expressible in

RATN form; in particular, there are many cases in which we simultaneously enter more than one system, as in the example given in section 3.1 of this chapter. Winograd points out that his PROGRAMMAR parsing procedures have the same computational power as Woods' RATN. The question then is, which is the clearer. Winograd argues against the RATN on three grounds, of which the most cogent concerns "demons". These are procedures which interrupt the parse process whenever they occur, and try to do some special parsing; an example is the special co-ordinate conjunction demon, run whenever "and" is met. Winograd observes that to represent a demon in a RATN would necessitate a special arc on every node.

Woods claims (1969) that one advantage of the RATN formalism is that one can as naturally run the network to produce as to analyse utterances. This is probably not true; certainly the RATN network in LSNLIS would be quite unsuitable for producing utterances because the conditions on arcs are all expressed in syntactic terms with no immediate relation to what the utterance is supposed to mean. In general, if the conditions on arcs become very complex, as they inevitably must in a system which takes account of meaning and as systemic grammar is designed to do, the RATN representation looses its perspicuity and so its principal advantage.

CHAPTER 4

A SYSTEMIC FUNCTIONAL GRAMMAR

1. Systemic grammar

1.1 Introduction

This section outlines the theory of grammar upon which the program is based. Almost the whole of it derives from the group of linguists at University College, London, who developed "systemic grammar" from the prototype set out in Halliday (1961). Much of the work on systemic grammar is unpublished, being transmitted orally and by mimeo sheets; the published work upon which the program rests includes Halliday (1967-8, 1970), Hudson (1971), Huddleston (1972), and to a limited extent Sinclair (1972). Winograd (1972) represents a development in a rather different direction. His systemic analysis was complemented not by structure-building rules (Hudson 1971) but by program procedures including semantic and contextual tests.

The present program also uses simplifications of Halliday's {TRANSITIVITY} systems proposed by E K Brown, though he is in no way responsible for the use made of them, and suggestions made by Hudson (mimeo 1972a, 1972b).

The technical language of systemic grammar is still developing, and in any case is less familiar than that

of transformational grammar. The next section provides a brief definition of the terms and formalisms used.

1.2 Overview and examples

Like a transformational grammar, a systemic functional grammar generates well-formed utterances: that is, it specifies the grammatical items which may occur, and it specifies their occurrence and combination by rules which tell us how an item may be well formed. Transformational grammar is in principle direction-independent. It is

no better adapted to produce well-formed utterances than it is to assign structure to a given utterance. Generative systemic grammar, on the other hand, is direction-dependent: as we shall see shortly, the rules are productive, not analytic, and are not reversible. Consequently we shall stop using the direction - independent term "generative" to characterise the grammar, and shall instead call it "productive" to emphasise that we are talking about constructing utterances.

Systemic functional grammar is a specification of the clauses into which grammatical items may be analysed together with rules for using the classification of an item to derive the immediate constituent structure of it. This chapter describes the classificatory apparatus, and the next explains the derivation rules. However, we repeat here in summary form the principles explained in the next chapter, in order to outline the grammar as a whole.

A uniform cycle of operations is responsible for the construction of each grammatical item. A symbolic representation of the item's meaning, accompanied by a partial grammatical description, is given to the constructor procedure appropriate to that type of item. The constructor completes the description by reference to the symbolic representation and perhaps to other criteria: it is in respect of knowing how to complete the partial

description that each constructor is a specialist for its particular type of item. From this completed description the rules of the grammar set out in chapter 5 now derive a partial or perhaps complete description of each immediate constituent of the item, ready for the next application of the cycle. The scope of the present grammar extends from clause items to word items, as explained in the next section. Consequently the productive process starts with the semantic representation and partial description of a clause item, and stops when each word has been assigned a grammatical description. For the moment we say nothing more about the procedures which supply the clause representation and partial description, and nothing about the procedures which select the right word in each case. These procedures will be fully described in chapter 6.

As an example of the cycle we may consider the construction of "us" in "among us". The partial description provided to the constructor includes the information that the item is a noun-group, {Ng}, and, being governed by a preposition, is an {Object}. The constructor examines the symbolic representation of the entities to which "us" must refer, and thereby is enabled to complete the grammatical description with the information that the item is plural, {Pl}, animate, {An}, and so on. It determines that the item is to be a pronoun, {Pron}, by means which are explained in chapter 6 section 7.6.

The constructor now applies the rules of the grammar to the completed description of the noun-group, and derives what is in this case a complete description of the single immediate constituent of the noun-group, namely the word "us".

This chapter is concerned with the elements of the grammatical description supplied to, and completed by, each constructor. The elements are called "features", and are explained in the next section. This section is concluded with a rather more detailed examination of the use made of the completed feature description of each item.

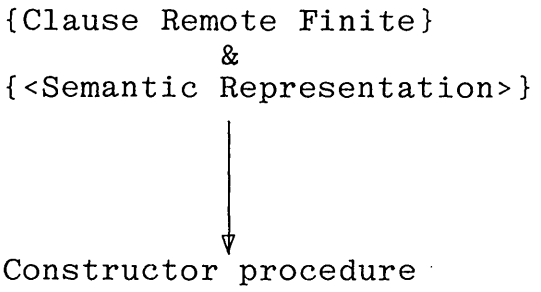
The productive cycle sketched a moment ago is divided into three major phases. Suppose we are constructing the clause:

"Bill kicked the ball".

The clause constructor is supplied with the information that the item is a clause, {Clause}, with past tense {Remote}, and finite, {Finite}. It is given other features too, which we ignore here. The first phase of the cycle requires the constructor to examine the representation of the clause's meaning and so to complete the feature description with details of the clause transitivity (see further section 4.1 of this chapter.). In this example the verb is transitive and the surface object explicit, all of which for the moment we abbreviate to

{Gtrans}: details of the full analysis of transitivity appear in section 4.1 of this chapter.

Events thus far may be diagrammed:



Phase 1: {Clause Remote Finite Gtrans}

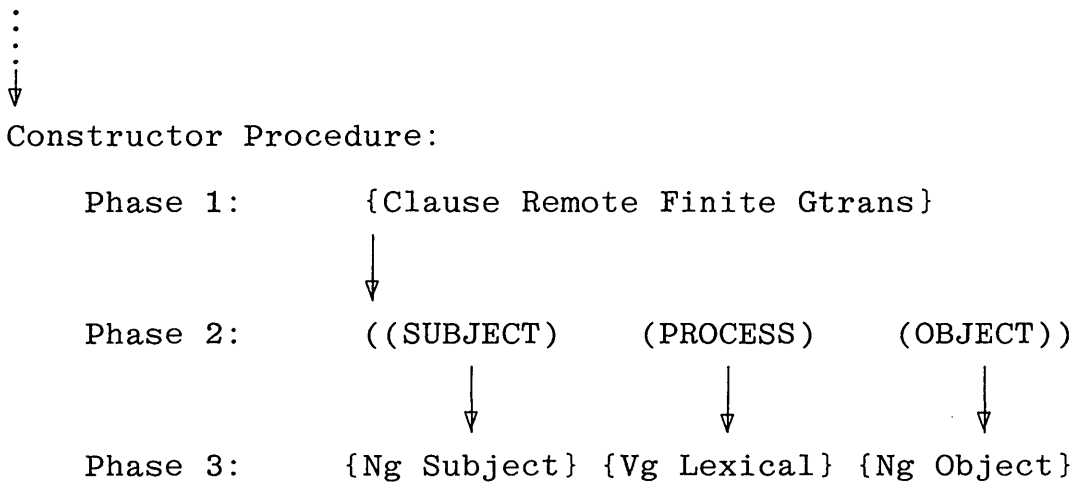
In this diagram the square brackets { } enclose a list, as before. The angle brackets indicate the presence of an internal computer representation of the item in question; so in this example the <Semantic Representation> indicates the presence in the list of a pointer to an address in the computer's store where the internal code is to be found representing the semantic information to be conveyed.

{Clause Remote ---}. This yields a description of each immediate constituent of the clause, telling us everything about the constituent which depends upon its being in a particular place in the string of constituents and having particular relations to other constituents. The description tells us what function, or functions, the constituent performs in the clause. The term "function" is more fully explained in section 2 of the next chapter: for the moment it may be taken simply to refer to a job being done by the constituent. In the present example we shall say that the first constituent, "Bill", has, among others, the function SUBJECT whereas the third, "the ball" has a different one, namely OBJECT. So far, then, we have derived a functional description of each constituent. The description summaries the environmental constraints upon the constituent, and we shall therefore call it a syntagmatic description, in contrast to the classificatory, or paradigmatic, description in terms of features. (Hudson 1971: 39 ff). The first and second phases may be diagrammed:

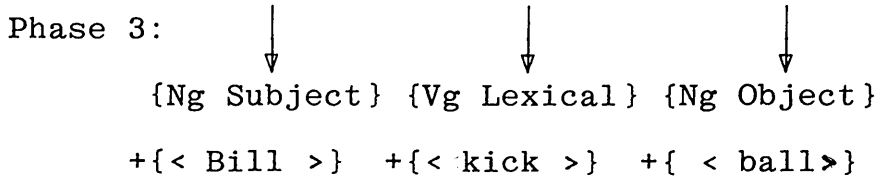
Constructor Procedure:

Phase 1:	{Clause Remote Finite Gtrans}
	↓
Phase 2:	((SUBJECT) (PROCESS) (OBJECT))

The third and final phase of the cycle is to derive for each immediate constituent a partial or complete feature description from each function description. For example, just as certain features of the clause item enabled us to deduce the presence of an immediate constituent with the function SUBJECT, so now the SUBJECT function enables us to characterise the associated constituent as a noun-group, {Ng}. Of course, many features of the noun-group are not determined by the SUBJECT function. As we saw in the previous example, the noun-group constructor has to decide for itself whether the noun-group is plural, animate, and so on. The three phases may be diagrammed:



The clause constructor now associates the appropriate semantic representation with each feature description



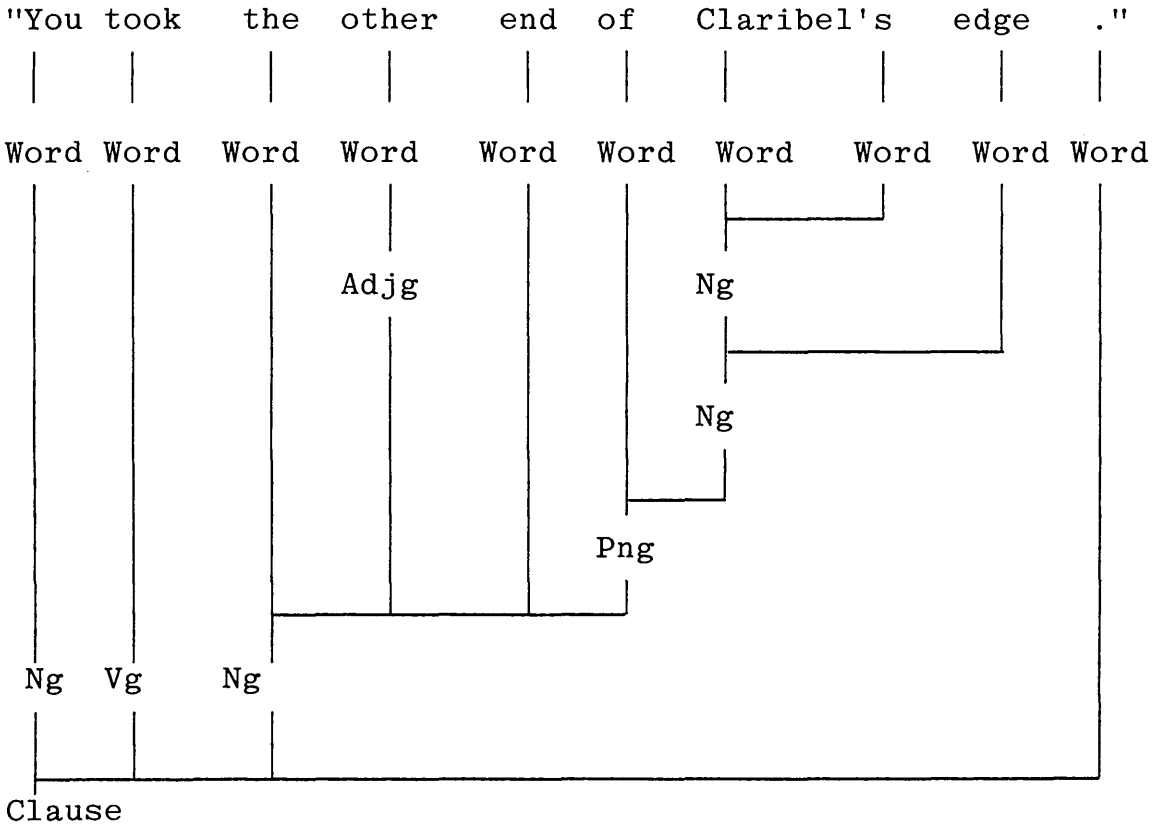
We are now ready to repeat the productive cycle upon each immediate constituent of the clause, invoking noun-group and verb-group constructors as appropriate.

Enough has been said here to show why we want the features set out in the remainder of the chapter. We need to be able to give each grammatical item a feature description which is adequate to permit the derivation of a functional description of each immediate constituent. The two examples given here have been much abbreviated and simplified for the sake of clarity. Further examples which fully reflect the operation of the grammar appear in the next chapter, and Appendix B contains a worked example which displays not only the features and functions of the grammar but also some of the more important internal variables of the program procedures.

4.1.3. Grammatical items

The classificatory apparatus of the grammar classifies grammatical items. An item is an unbroken word string,

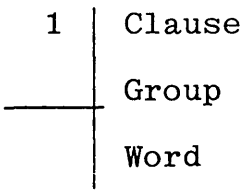
of length one or more words, and is a clause, a group, or a word. A clause is an item which conveys an attribution, a state, or an event; it therefore comprises at least a lexical main verb and, unless imperative, at least one nominal participant. A group is a constituent of a non - coordinate clause, not itself being a clause or a word: the four types of group, noun-group, verb-group, preposition - group, and adjective-group, are described in detail later in this chapter, and figure in the example below. The term 'word' is taken to be well-defined in its conventional sense, but is extended here to include punctuation marks and the genitive morpheme " 's ". The following constituent structure diagram exemplifies all the kinds of item mentioned:



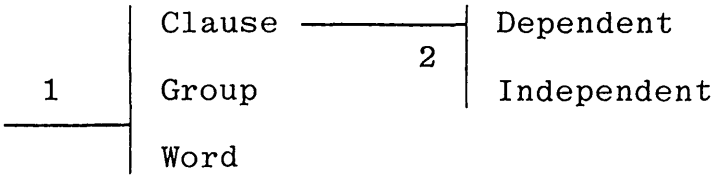
4.1.4 Features and systems

This section explains how the grammar defines the classes to which an item may be assigned. An item classified as belonging to a class is said to have a feature, and we write the feature name in square brackets. The last example displayed the constituent structure of an item which had the feature {Clause}.

A feature assigned to an item is drawn from a set of simultaneous alternatives. For example, every item must have just one of the features {Clause}, {Group}, or {Word}. Such a set of simultaneous alternatives is called a system, and is written:

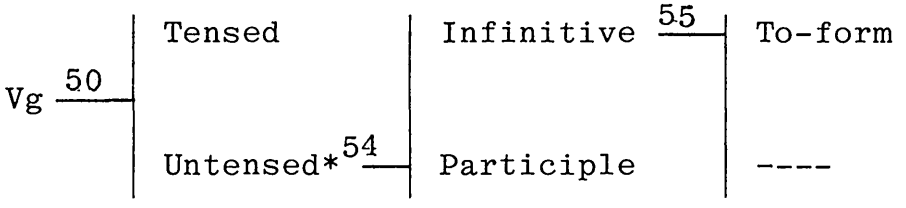


The simultaneous alternatives are written to the right of a vertical bar, and the system is numbered for easy reference. The selection of one alternative usually determines what further system, or systems, may be entered. For example, if the item has the feature{Clause}, we need to know whether it is {Dependent} or {Independent}. We show that {Clause} is the entry condition to this further system as follows:



System 2 does not apply to items which are {Group} or {Word}, and so we say that system 2 makes a 'more delicate' distinction than system 1.

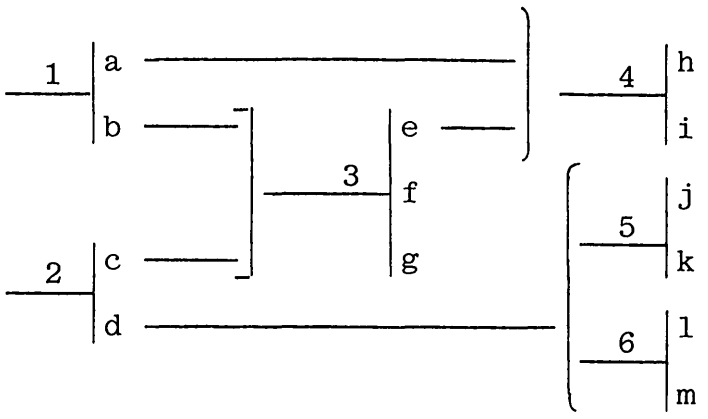
Sometimes we want to show that one alternative is the normal, or unmarked, one, to be selected in the absence of indications to the contrary. We show this by putting an asterisk on the unmarked choice, or, if we don't wish even to name it, by writing simply '---'.



We are concerned only with the formalism here: the significance of the systems in the example is explained in the section of this chapter on the Verb-group.

Additional expressive power is gained by using square OR brackets and round AND brackets. Facing right these mean 'select one' or 'select all' respectively. Facing left they mean that one, or all, respectively of the entry conditions must be satisfied.

So in:



system 3 is entered if the item is {b} or {c}, system 4 is entered only if the item is both {a} and {e}, and systems 5 and 6 are entered simultaneously if it is {d}. The right facing OR bracket is simplified to the vertical bar which we have met already.

It is probably obvious that there may be several ways of arranging a systems network. For example, we might make it a rule that every system was binary. The arrangement selected depends partly upon considerations of parsimony and simplicity of lay-out: we want as few features as possible, but simultaneously as few lines and complicated brackets as possible, and these two criteria often conflict. In the present case a computer program embodies the grammar, and programming considerations sometimes make particular arrangements of systems desirable. In fact the program does not proceed through every network exactly as the layout suggests, because it is sometimes inconvenient to do so.

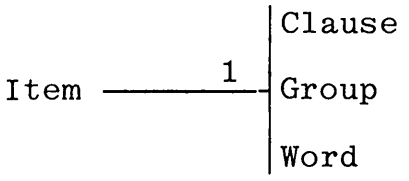
4.2. A systemic functional grammar

The remainder of this chapter presents the systems of a systemic functional grammar which guides the production of all the English output by the program. The grammar produces by no means every well- formed English utterance: it is very greatly simplified and generally captures only those distinctions which are necessary for the program's output. However, it is designed with the intention of being expansible by the addition of more delicate systems on the whole, rather than by melting and recasting. A complete systemic grammar, like a complete transformational grammar, would be complex and hard to test. The present grammar is crude and in parts ad hoc, but has at least been thoroughly tested within its planned scope.

In section 1.2 we saw that an item's functions may determine some of its features, SUBJECT requires us {Ng}. It will shortly become apparent that this allows us to state environmental restrictions upon systemic choices very economically. To quote an example from Hudson (1971) we need no special rule to kill non-finite forms of modal verbs, because {Finite} is an entry condition to the {Modal} system, and {Finite} depends upon the item having a particular function in the clause. So the question of modal verbs being non-finite never arises.

4.3 The {TYPE} system

The first system in the network is:

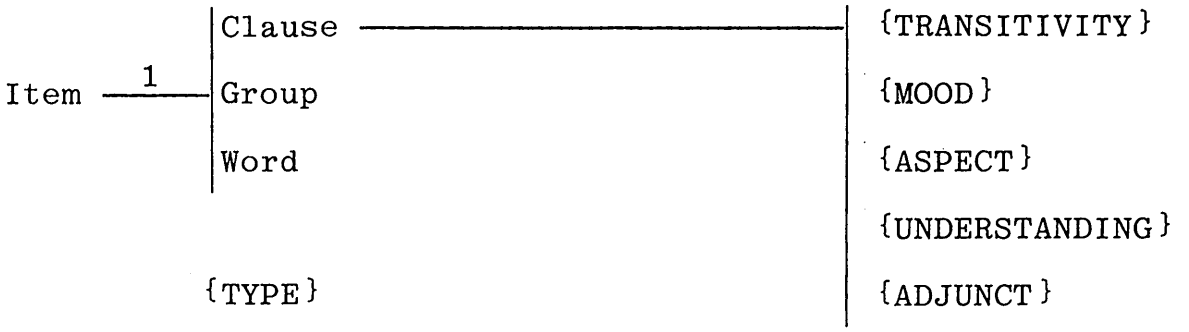


{TYPE}

As we saw earlier, every item is a {Clause}, a {Group}, or a {Word}. This formulation integrates the scale of rank (Halliday 1961, 1967-8) into the systems network. The scale of rank was the scale (...Clause - Group - Word - Morpheme...): items had a place in this ranking, but could be 'rankshifted', so, for example, a 'rankshifted' clause could do the job of a group as the subject of a sentence. Hudson (1971) expunged the scale of rank from the theory, and we have here adopted his more parsimonious version.

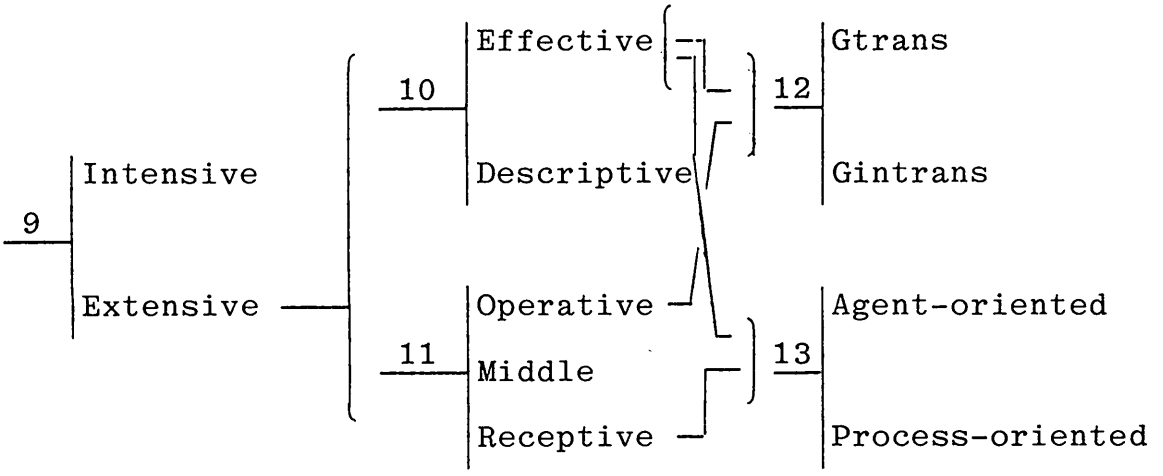
4.4 Clause systems

This section explains the clause systems network. It falls into five independent sub-nets, {TRANSITIVITY}, {MOOD}, {ASPECT}, {UNDERSTANDING} and {ADJUNCT}, which we can summarise as follows:



4.4.1 The {TRANSITIVITY} systems

The {TRANSITIVITY} systems of the clause are taken from Halliday (1967) via a formulation of E K Brown (p.c.). They are simple, and do not pretend to capture all the distinctions to be made. They do not follow Halliday's later ergative analysis (1970), which attended primarily to what was affected by the action and then to who the causer was. Instead they distinguish the participant roles of Actor, the participant doing the action, Initiator, the participant causing the action, and Goal, the participant at which the action is aimed. The systems are:



System 9 distinguishes extensive from intensive, or attributive, clauses. It happens that the output of the program includes no intensive clauses, so that the problems of this type of clause are not explored.

Following Halliday's (1967) terminology, system 10 distinguishes effective actions, which may be directed at a goal and in which Actor and Initiator inhere in one participant, from descriptive, in which there is no Goal participant and the Actor and Initiator are separable. So examples of effective clauses are:

"She slimmed her hipline"

"She slimmed"

"Her hipline was slimmed"

and of descriptive are:

"I started the engine"

"The engine started"

"The engine was started by the A.A. man"

System 11 distinguishes Operative, where the Initiator is surface subject and is distinct from the Actor when permissible, as:

"She slimmed her hipline, I started the engine"

from the middle, in which all participants are combined in the surface subject:

"She slimmed, the engine started"

and from the Receptive, or passive, in which the surface subject is not the Initiator but the Goal in {Effective} clauses, and the Actor in {Descriptive} ones:

"Her hipline was slimmed, the engine was started by the A.A. man."

An {Effective Operative} clause may have an explicit or an implicit Goal. If {Gtrans} we have:

"You have won the game."

if {Gintrans}:

"You have won".

System 12, like the rest, classifies the clause, not the verb; it marks the distinction between a clause with, and a clause without, a particular participant, not a distinction between a verb which can, and a verb which can't, govern a direct object. It applies only to {Effective Operative} clauses since it concerns the Goal participant in non-subject position, so {...Gintrans} is incompatible with {...Middle}.

System 13 makes a distinction which does not apply to all {Effective Receptive} clause, and is not made in the program's output. Halliday introduced it to distinguish:

"...the clothes were washed"

from:

"The clothes washed (clean,easily)".

In both examples the Goal participant is surface subject, but the form of the verb, and the consequent 'orientation' of the clause, is different in the two cases. There are many verbs which do not permit the {Process-oriented} clause, a fact which should probably be explained in terms of more delicate distinctions. No verb used by the program permits the {Process-oriented} form: a game commentary is in terms of who-did-what, and so the {Agent-oriented} form is normally more suitable.

4.4.2 {MOOD} systems

The {MOOD} systems classify the kind of job the clause is doing, and define the consequences especially for the form of the verb constituents. In the present grammar the network also includes some clause-structural classifications (systems 27, 28, 29) which are dependent upon the clause being non-finite, but are not otherwise {MOOD} systems.

The MOOD network appears on the next page. The names of the features have been chosen to make the systems self-explanatory as far as possible. System 14 distinguishes clauses which are not constituents of any other item except a coordinate clause from those which

are dependent. The present {MOOD} systems allow only an {Independent} clause to be {Imperative}, {Interrogative}, or {Declarative}. Clearly commands, questions, and statements can be embedded in {Dependent} clauses and must be distinguished from each other in that circumstance. However, the program produces no such embedded clauses, and so no attempt has been made to produce a grammar capable of accounting for them. Instead we have systems which really analyse not the form of the independent clause but the type of illocutionary act, and which rest on the assumption that type of act and form of clause have a unique correspondence.

System 15 distinguishes the imperative, which the program does not use, from the indicative. Systems 15 and 16 are done the way they are, rather than in a single 3-way system, because imperatives are not indicative, whereas statements and questions are. Compare 'You are good' with '(You) be good'. Indicative clauses must be questions or statements, system 16, and we know that all questions will be polar, yes-no, questions, system 17. No attempt has been made here to write a grammar of question clauses; such a grammar would be complex, and would have to be properly related to the systems classifying relative clauses and embedded questions:

"I asked her whether she came there often"

An indicative clause, or a finite dependent clause, system 18, may be modal and may be past tense, systems 23 and 25. Imperative and non-finite dependent clauses cannot be modal or have past tense in their own right, though they may fall within the scope of a modal or a past tense and be so interpreted:

"Having seen the joke, you would be rude not to chuckle a bit"

"Having seen the bull, you were ill-advised to flaunt your pink socks".

Dependent clauses are, as we have seen, finite or non-finite; if finite they are relative:

"(the man) who sold you a gold brick"

"(the man) whose face you instinctively trusted"

or bound, system 19:

"Although he had 101 convictions (he loved children and dogs)".

Other binders include 'if, when, because, after,...' and so on. Relative clauses have peculiarities which systems 22, 26, 300, 301, and 302 try to capture. The relative word comes at the front of the clause, regardless of its participant role in the relative clause. It may be the object of a preposition, {Preprel}: if it is, the preposition may be fronted, as in :

"(the damage) for which they are responsible"

or left dangling at the end:

"(the damage) which they are responsible for".

If the relative word is not governed by a preposition, the clause is {Nomrel} and we note what participant roles the relative pronoun plays. It may be the Actor:

"(the men) who were marched up the hill and down again"

Initiator:

"(the Duke) who marched 10,000 men to the top of the hill"

or Goal:

"(the car) which I pushed".

System 26 marks the case in which the relative word is explicit. Very often we leave it out, but only if it is not governed by a fronted preposition and is not the subject:

* "the insults with I won't put up"

* "(you have met the professor) mends alarm-clocks".

The systems here capture the first proviso, but the second cannot be shown in the same way because it depends upon a prior determination of what participant roles are realised in the clause. This decision would be taken by the building rules operating upon the features derived from the {TRANSITIVITY} systems. So the suppression of the relative word can be done only after the

subject has been decided; the relevant rules would be simple to devise, but have not been written because it happens that the program never has to construct a relative clause whose subject is a relative word.

System 27 distinguishes two types of non-finite dependent clause, infinitival:

"(he only does it) to annoy"

or participial. The latter may be {Ing}:

"(I won by) smashing a Bechstein in thirty seconds"

"(We camped early,) planning to start at dawn"

or {En}:

"He collapsed, exhausted by the climb".

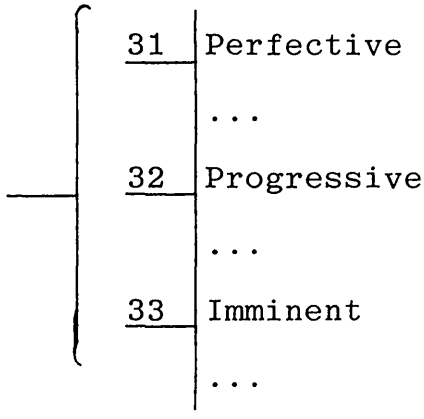
A {Participle} clause may be simply adjoined, as in the last examples. But the marked choice in system 29 applies to the case where such a clause is introduced by a binder:

"Although exhausted by the climb, (they at once began the descent)."

"Although denying everything, (they agreed to refund the deficit)."

4.4.3 {ASPECT} systems

The {ASPECT} systems are simple:



System 31 distinguishes Perfective

"(I'd like) to have swum the Hellespont"

from unmarked aspect

"... (but not) to swim it"

System 32 marks Progressive:

"... to be swimming it (would be miserable)"

and system 33 Imminent.

"... to be going to swim it (would be the worst)".

The choice is simultaneous in each of the three systems, since aspects may be combined fairly freely. Some rather complex combinations are excluded from the dialect of most people, but we do not try to capture these uncertain rules in the grammar.

4.4.4 UNDERSTANDING systems

4.4.4.1 Introduction

Grammatical items sometimes share, or 'understand', constituents elsewhere in the context. The context may be the immediate clause or sentence, it may involve less or more remote preceding exchanges, and it may even extend to factors outside the scope of any simple grammatical analysis, such as who the speakers are and how familiar they are with the subject matter; intimates talk shorthand. Isard (forthcoming) formalises rules governing the maintenance and loss of a modal context or time-reference in conversation, and Longuet-Higgins (Isard & Longuet-Higgins 1973) did the same for certain incomplete questions such as the third line in the following:

("Will Bob come after Al?")

("Yes.")

"Just after?"

The question "Just after?" requires completion by "Will Bob come (just after) Al?", which it must understand from the first question, though the program itself does not work in quite that way. Compare Winograd 1972 p.13.

In the present grammar attention is confined to clause items which understand nominal arguments of the main verb. The assumption is made that a complete well-formed indicative clause has at least a lexical verb and a subject; it also has an object if the verb is transitive and active and cannot be left without an object. Thus a clause is

{Understanding} if one of the mandatory constituents is not explicitly present in the item. Three rather different types of clause are accounted for on this basis. We first explain these examples and then move on to examine problems and alternatives.

The first case is that of dependent non-finite clauses with participial verb. The subject of such clauses may be omitted as

"(I began the game by) tripping the referee"

in which case the clause will be said to understand its subject. For reasons set out in section 5.2 of this chapter, we do not regard possessive determiners such as "his" and "Bill's" in

"I questioned John about his (Bill's) buying a bike."

as constituents of the participial clause. Consequently such clauses too are regarded as {Understanding} their subject. The problem for a productive system is to decide when the subject must be made explicit, either in the form of a determiner as in the last example or as "Bill" in

"I told John about Bill buying a bike".

The rule used by the present system is that the subject may be left implicit provided it is the same as the subject of the main clause. This rule is, of course, too simple since examples come readily to mind where

it fails. If someone is congratulated for doing something, he was responsible for it, so in

"I congratulated him on the winning the sack-race"

the winner was he. A more complicated example supplied by S D Isard, is:

"The Surgeon-General believes that smoking causes cancer"

In this case both the subject of the main clause and, presumably, anyone else at all is believed liable to cancer if he smokes. The criterion by which the smoker may be left implicit in this case is not clear, and the present system makes no attempt to propose an adequate rule.

The second case is that of dependent relative clauses, in which the relative pronoun may sometimes be left out. The present grammar accounts for the ommission of the direct object of the verb, as in

"(the square) you took first"

by saying that the relative clause understands the goal constituent, which would otherwise appear as "which"

"(the square) which you took first".

However, it must be pointed out that the grammar is not consistent in its treatment of implicit relative pronouns. A relative pronoun governed by a preposition, as

"(the horse) which she rode on"

can be left out

"(the horse) she rode on".

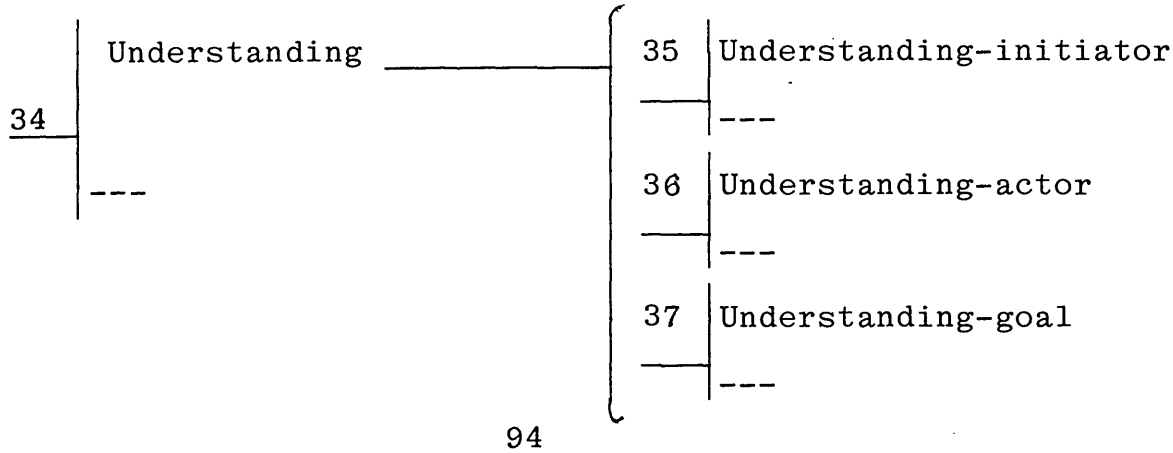
The present grammar does not use the {UNDERSTANDING} systems to account for this, although clearly what is sauce for the direct object should be sauce for the prepositional object too. The reason for this inconsistency is that the grammar does not include a comprehensive analysis of potential nominal roles in the clause and there is therefore no properly motivated way of referring within the {UNDERSTANDING} systems to the object of a preposition. An ad hoc addition to the grammar was felt to have no advantage over the present arrangement.

The third case is the most interesting. In

"The Keystone Cops rushed in, tripped over the rope, and fell in a heap"

the second and third co-ordinated clauses understand their subjects. As in the previous two cases, the items concerned here, "tripped over the rope" and "fell in a heap", are not complete and well-formed clause items in isolation.

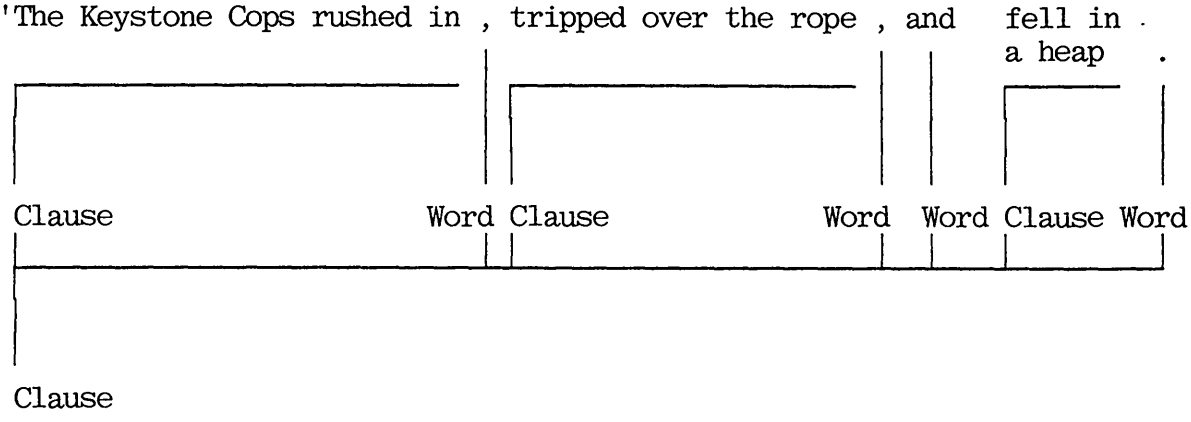
The systems used for these three cases are:



System 34 marks the clauses we are concerned with, and systems 35 - 37 specify which participants are understood. For example:

" ...(and) tripped over the rope"

is {Understanding Understanding-initiator Understanding-goal}, and the constituent structure of the sentence is, in summary:



4.4.4.2 What an {Understanding} item understands

The {UNDERSTANDING} systems given here apply only to {Clause} items, and only to those nominal constituents which normally have to be present in a clause for it to be well formed. They thus have nothing to say about clauses which understand other constituents, whether mandatory or optional, nor about items other than clauses which may understand constituents. We shall shortly return to this subject, but first consider the question whether an {Understanding} clause understands a word or a meaning. In the last example, the clause "tripped

over the rope" understood its subject, "the Keystone Cops", and the clause seems to mean exactly what "the Keystone Cops tripped over the rope" means. This might encourage one to think that an item which understands a constituent must be interpreted as though it were completed by the insertion of the understood word-string. However, even within the present very limited grammar this view is hard to sustain where indefinite phrases are understood. The usual rule in English is that once an indefinite noun group has been used, its referent is subsequently denoted by a definite noun group; thus in

"A man came in and he calmly sat on my hat"

"he" refers to the referent of "a man". So when a clause understands an indefinite constituent, as the second clause in

"(Someone came in and) sat on my hat"

we cannot say that this example is equivalent to

"Someone came in and someone sat on my hat"

since such a sentence would be taken by most people to imply at least some doubt whether the entrant and the sitter were identical, whereas in the original there is no such doubt. The present grammar ignores the understanding of optional clause constituents, but examples of these proposed by S D Isard in conversation reinforce the suggestion that what a clause understands is not the word-string but the meaning of the understood constituent.

4.4.4.3 Accounting for understanding.

Nothing further is said here about the first two kinds of {Understanding} clause, that is about examples such as:

"He began by telling a shaggy-dog story"

"The shaggy-dog story he told first".

It is assumed that the rules permittting such examples are peculiar to each type, as outlined already. However, the rules proposed to account for the third type of {Understanding} clause are probably more general: they are intended to apply not only to clauses which understand subject or object, but also to clauses which understand other constituents, and to other types of item which understand constituents. The discussion here, though, is confined to types of case which the program actually produces. In accounting for:

"The Keystone Cops rushed in, tripped over the rope, and fell in a heap"

two rules are suggested.

1. An item may be {Understanding} only if it belongs to a coordinated set.
2. An {Understanding} item may understand only an immediate constituent of another member of the set.

By rule 1, coordinated clauses may be {Understanding} without regard to the nature of the coordinating conjunction. In:

"(She knew, or) said she knew, ---"

"(She didn't know, but) said she knew"

the clause "said she knew" understands its subject.

Rule 1 does not apply to participial or infinitive non-finite clauses as

"We watched her digging the garden"

"We asked her to come to supper"

nor, as we saw, to finite relative clauses, which seem to be a special case. It does, however, prevent:

*"He said that was coming"

and the rendering of

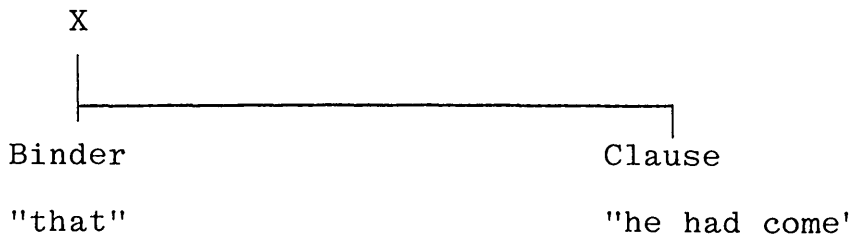
"I shall say that I shall come"

by

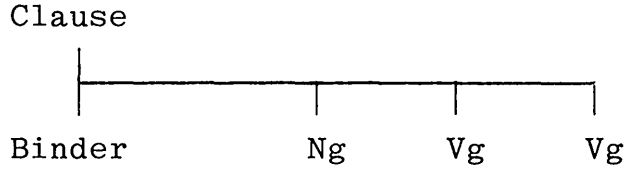
*"I shall say that come".

Rule 2, of course, depends for its interpretation upon assumptions made in the grammar about constituent structure. The assumptions made in the present grammar are made explicit in this and the next chapters, but might not be agreeable to everyone. Alternatives are considered in the section, together with some reasons for the suggestion made here. The constituents of a clause are taken to be noun-groups realising subject and object, verb-groups realising lexical and grammatical verb functions, binders which are usually words, and adjuncts of various types. Binders are treated as clause constituents because it is otherwise necessary to postulate some grammatical item of which the binder and the clause

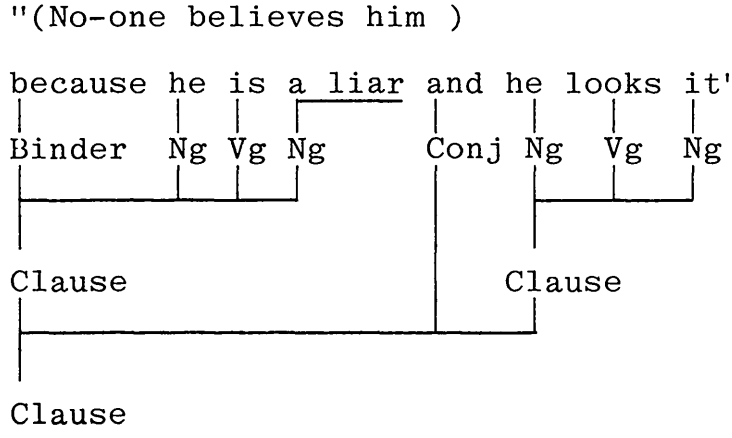
it introduces are constituents:



and the present alternative:



makes unnecessary the postulation of such an intervening level of structure. Binders can thus be understood where subordinate clauses are co-ordinated:



The clause "he looks it" is a subordinate clause which understands its binder from the coordinate preceding clause. It might have understood its subject as well:

"--- because he's a liar and looks it".

We have already noticed an example of a clause understanding a grammatical verb constituent. The lexical verb may also be understood, in what J R Ross called "gapping":

"(You have the Glen Morangie and) Bill the Glen Fardas"

So far no reference has been made to the order of the items which respectively understand and have constituents understood. The rule governing the order seems to be, in the case of clauses only, that

3. A clause which understands a constituent, must follow the clause of which the understood item is a constituent, if the constituent is the lexical verb or precedes it.

Rule 3 thus prevents

*"You the Glen Morangie and Bill have the Glen Fardas"

If the understood constituent follows the lexical verb of its clause, the understanding clause may precede:

"You start (, and I'll finish the bottle)"

in which "you start" understands the noun-group "the bottle". In practice this construction is often felt to be somewhat uncomfortable; "you start" might thus be felt to be

complete by itself and not to understand anything, or alternatively the clause "(and) I'll finish" might be parenthetical:

"You start, and I'll finish, the bottle"

Both of these proposals are plausible, but there seems to remain a place for the account favoured here. Similar problems may arise when an optional adjunct follows the verb in the second clause:

"She came in and did the flowers later".

We state here that this sentence comprises two clauses, of which the first, "she came in", understands the adjunct constituent "later" from the second. An alternative account is considered in the next section, but notice here that because the adjunct is syntactically optional in the first clause, it is possible to argue that "she came in" doesn't understand anything. However, her "coming in" seems to have happened "later", at the same time as she "did the flowers," and so the interpretation given here seems most natural.

In concluding this section it must be emphasised that these three rules are permissive only, and do not require that any item understand any constituent. Nor do they offer much guidance to a procedure which is seeking to assign structure to a given input; that problem is not our business now, but in passing we recommend the rules mentioned by Winograd (1972 pp 71,149) for assigning

structure to coordinate items as offering powerful help, since rule 1 confines {Understanding} to a constituent of a coordinate item.

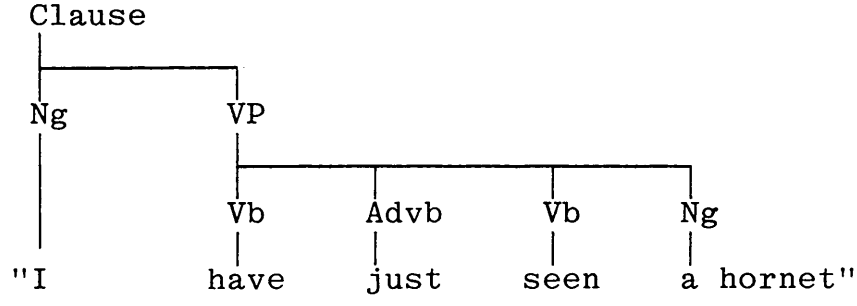
4.4.4.4 Alternative accounts

The problem being considered here is how to account for the way coordinate clauses can sometimes leave constituents out when the context contains copies of them. The account given in the previous section suggested that grammatical items which are otherwise normal are marked as {Understanding} the missing constituents, which must appear as immediate constituents of a coordinated item. We shall mention two alternative accounts in this section, though more briefly than they deserve.

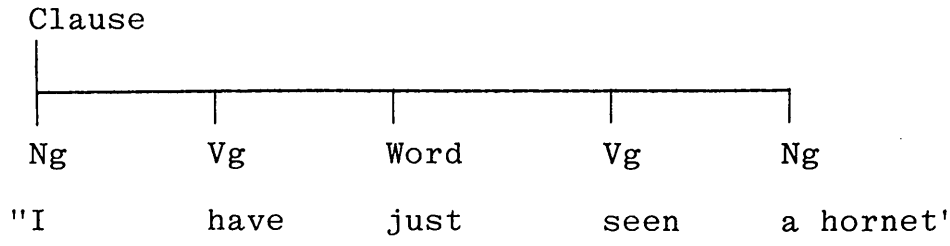
One alternative involves the postulation of a Verb-phrase item as an immediate constituent of a clause. The structure of:

"I have just seen a hornet"

might then be:



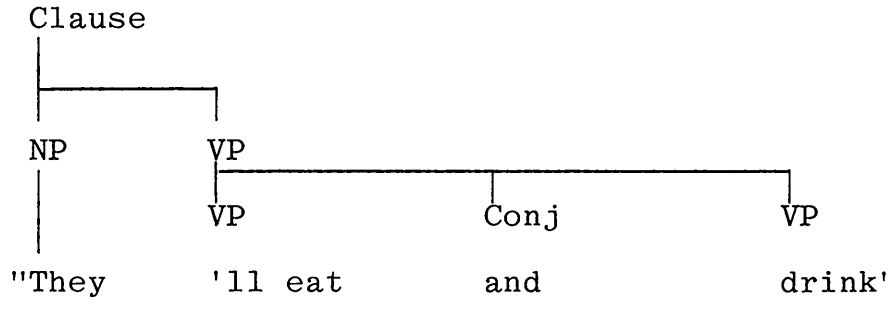
and not, as at present proposed:



The presence of a VP constituent which includes the entire complement of a sentence makes certain cases easier to explain. The structure underlying:

"They'll eat and drink"

is now assumed to be:



in which the second VP understands the modal future "will" from the first, whereas before we said that the structure comprised two clauses, of which the second, "drink", understands subject and modal verb from the first. The explanation in terms of VPs is more economical. The postulation of a VP also assists us to explain the difference between:

"She'll return and do the flowers later"

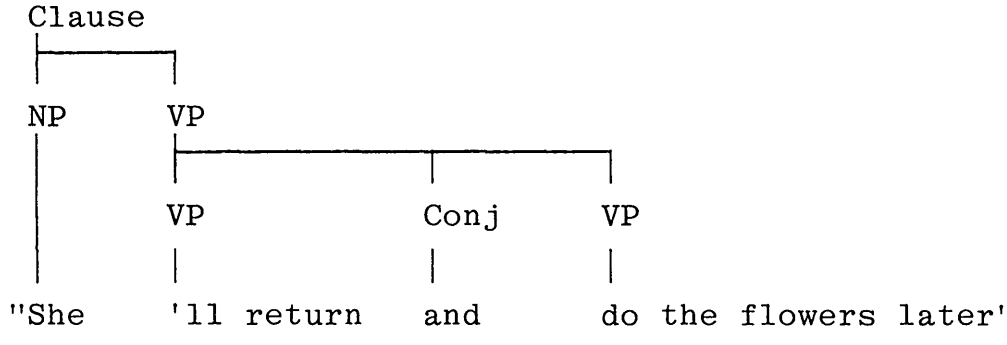
and

"She'll return, and will do the flowers later".

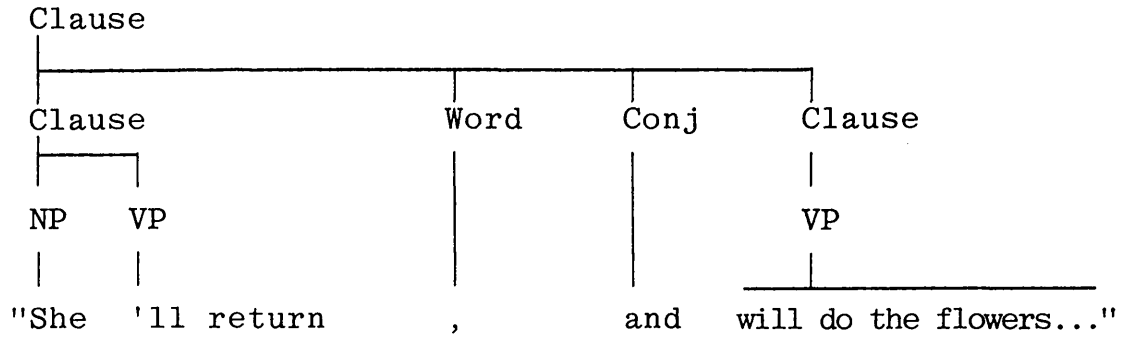
In the first case we observe that "later" is most naturally taken as modifying both verbs, and that "do" does not have an explicit "will" immediately before it. In the second case, "later" is most naturally taken with "do" alone, and "do" is preceded by an explicit "will". It is a matter for debate whether:

"She'll return, and do the flowers later"

is acceptable, but it would probably be agreed that the heavier the pause marked by the comma the less acceptable it becomes to omit "will". In the first example we might say that the structure is:



and the first VP understands "later" from the second, while the second understands "will" from the first. By rule 2 this understanding is permissible since immediate constituents of coordinated items are being understood. In the second example, the comma segments the sentence and marks the fact that the structure is now:



Now "will" and "later" are no longer immediate constituents of the items coordinated and hence cannot be understood by those items. In the absence of a VP item from the grammar, our explanation must rely upon the segmentation marked by the comma to rule out understanding of "later" by the clause "she'll return", and of "will" by "do the flowers later". The alternative account in terms of constituent structure and the application of rule 2 may seem better motivated.

However, the VP item has been excluded from the grammar, though at a fairly late stage of development. The reasons were that it interpolated a level between clause features such as {Modal} and the verbs which realise the appropriate functions, and it has to be regarded as a discontinuous item in questions:

"Did you kick Cock Robin?"

and in such negative contexts as:

"Scarcely had the kettle boiled..."

It would make it hard to account for gapping:

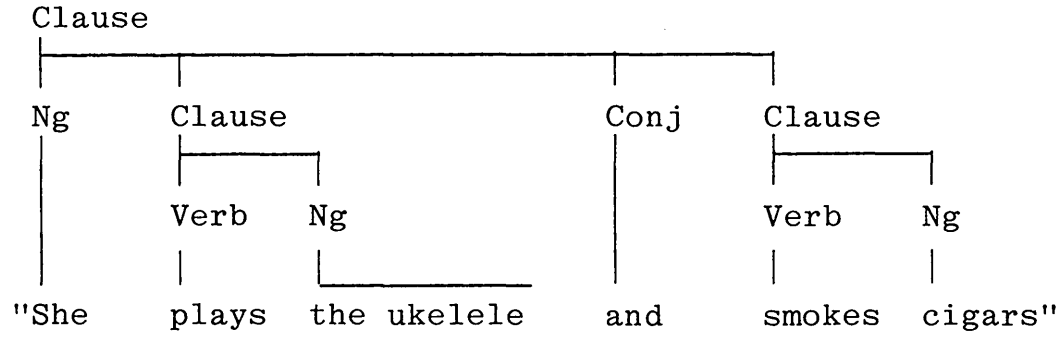
"Fred gave beer and Mary gin"

consistently with rule 2. It was eventually discarded.

The second alternative account is offered by Hudson (1972a). He proposes that the structure of:

"She plays the ukelele and smokes cigars"

is:



The essential feature of this proposal is that it 'factors out' the constituents common to the coordinated items and puts them on the left (or right) end of the string of coordinated items. This alternative has not been adopted for the following three reasons. First, taking "plays the ukelele" and "smokes cigars" as parallel items becomes less plausible the less closely together we understand them. In the current example playing and smoking are closely associated, but in:

"She's giving a recital and will return tomorrow"

giving a recital and returning are less closely associated and the natural intonation of the example reflects this.

In such cases we may wish our structural analysis to contain an item corresponding to the fully grammatical clause

"She's giving a recital"

such as the proposed alternative does not contain. It is debatable whether different structural analyses should be given to the first and the second of these examples: the present system has only one at its command.
Second, the proposal as originally made included no provision for marking the co-ordinated items with features to show what they understood. This is perhaps not a great difficulty, except that it could be overcome only by planning the entire sentence at the start and arranging the structure and features appropriately; the existing construction procedures remain unaware of the second clause, and that it understood its subject from the first, until they come to build it. This is a great practical advantage. The third objection has force only within the present grammar, and it is that the proposed alternative involves discontinuous items, as the "factors" are hitched to each of the co-ordinated items. This makes the rules complicated and hard to follow in operation.
There are, of course, discontinuous items in English such as

"The roses, he said, would be early this year"

and a full grammar would have to account for them. For the moment, though, an explanation of understanding which does not involve discontinuous items is to be preferred.

4.5 ADJUNCT systems

The ADJUNCT systems classify the material which may occur in a clause in addition to the immediate constituents which realise BINDER, PROCESS, and transitivity functions. The systems shown here make some elementary distinctions but are not meant as the basis of a full analysis: the options for Adjunct constituents are extremely complex, and are certainly beyond the descriptive capacity of a grammar as simple as the present one. We mention four problems in particular. First, for almost any position in the immediate constituent string of a clause we can find an adjunct which can occur there. Second, to complicate the matter even further, many kinds of adjunct are not bound to any particular site in the immediate constituent string, but may occur in one of several places as the semantics demand, and perhaps in accordance with delicate distinctions of emphasis and prominence which we have not formalised. The grammar contains no notation to handle this variability in a natural way. Third, there may be many adjunct constituents in a clause, particularly in colloquial English. The elementary grammar presented here allows only three adjuncts, all different, per clause; this is an adequate rule for the output of the program but is in no way general. The final problem is that an adjunct may convey one of a variety of meanings, and may convey a given meaning in one of a variety of syntactic structures. For example, a temporal adjunct modifying the clause may be conveyed by "hours" (Ng), "for hours" (Prepg), "too long" (Adverb-group), or in a variety of clauses. There is some

interaction between meaning, structure, and position in the clause, but there remains a great deal of variation which is motivated very delicately, if at all. For example, English, unlike German, does not order adverbial adjuncts by meaning.

> "We finished the party by the Scott monument at two in the morning"

is quite as acceptable as:

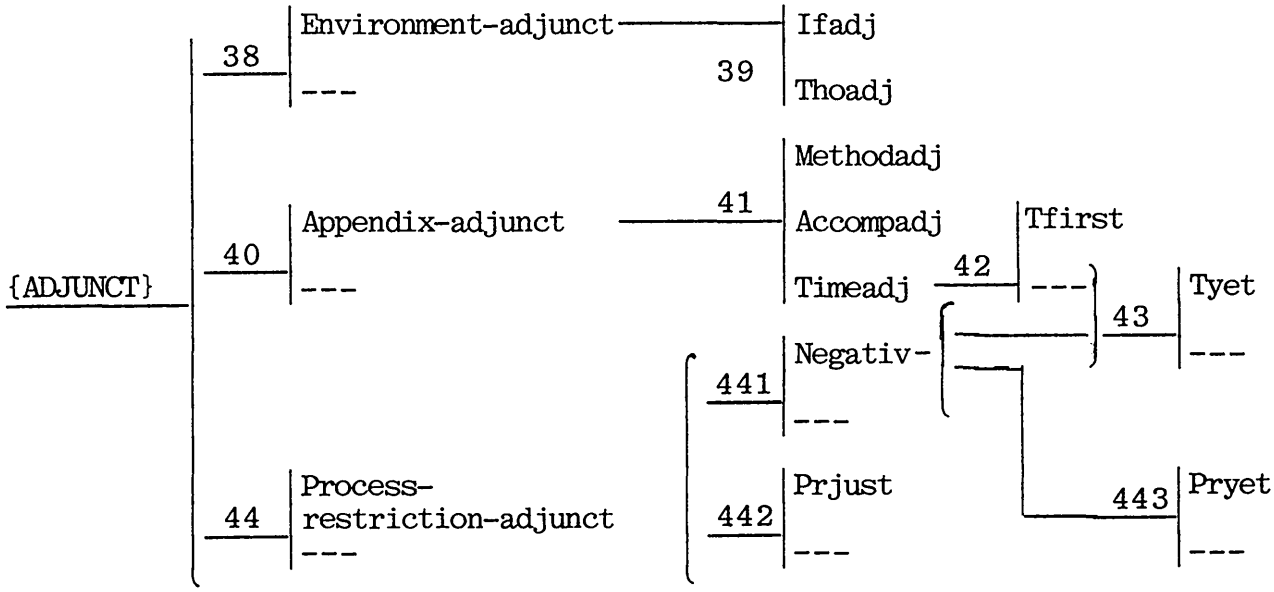
> "We finished the party at two in the morning by the Scott monument".

We often find that a short adjunct, perhaps an adverb, precedes rather than follows a longer one, to avoid getting lost in the absence of heavy emphasis:

> "They loaded the ship there in a hurry"
> * "They loaded the ship in a hurry there"
> "They loaded the ship hurriedly at Leith"
> * "They loaded the ship at Leith hurriedly".

An approach to solving these problems would require analysis of the range of meanings, structures, and locations in clause-structure of adjuncts, and of the way these three dimensions of analysis interact. It would require some enrichment of the apparatus of the grammar to cope with free variation of location within clause-structure, and a satisfactory representation of indefinitely many adjuncts within a fundamentally classificatory grammar.

The ADJUNCT systems in the present grammar divide adjuncts into three types according to their position in the clause. An environment adjunct comes first in the clause and establishes the context or circumstances within which the rest is understood. A process-restriction adjunct closely adjoins the finite verb and modifies it. An appendix-adjunct comes last in the clause, and is so named because it generally enlarges upon information already given. However, as we noticed a moment ago, there is no constant relation between adjunct function and adjunct location within the clause, and so the adjunct names used here are not always apt to the job being done by the adjunct: an example will be given shortly of an appendix adjunct which provides an environment. The adjunct names have been selected to give an idea of what the adjunct concerned is usually doing, but this part of the grammar is the least satisfactory of all and would need a complete revision to be of any real of any real interest.



System 39 distinguishes two type of environment adjunct: {Ifadj} marks a clause whose first constituent is a conditional clause:

"If you tweak my nose, my ears flap"

and {Thoadj} one whose first constituent is a concessive clause:

"Although he's Scottish, he dislikes whisky".

In both these examples the adjunct provides a background context for the remainder of the clause; in many cases such a context is essential and the main clause can't be understood without. So Isard's program (forthcoming), when asked out of the blue:

"Would you take 5?"

responds with a puzzled:

"Under what circumstances?"

An environment adjunct defines the circumstances, thus:

"If I took 4, would you take 5?"

Notice that a simple past tense requires a time reference which is often supplied explicitly by an environment adjunct:

"When Bill came, the party warmed up"

"After Bill's arrival the party warmed up".

System 39 does not distinguish a time environment adjunct because the program does not produce any clause with a time reference in first position in the clause. However, time

references provide an environment within which the rest of the clause is to be understood, very much as a conditional clause provides a context for the modal main verb.

The process-restriction adjunct comes close to the verb and is understood closely with it. The adjunct's exact position depends upon what grammatical verbs, if any, accompany the main verb and whether the main verb is a question. System 442 marks {Prjust} a clause which has "just" modifying the verb as:

"Bill has just come".

We notice that "just" resembles 'barely, scarcely,...' in clinging closely to the verb, and differs from words like 'soon' in this respect, even when conveying a time reference as it does in the program's output:

"...which you had just taken".

{Negativ} in system 441 marks a clause which is negatived by "not" with the main verb:

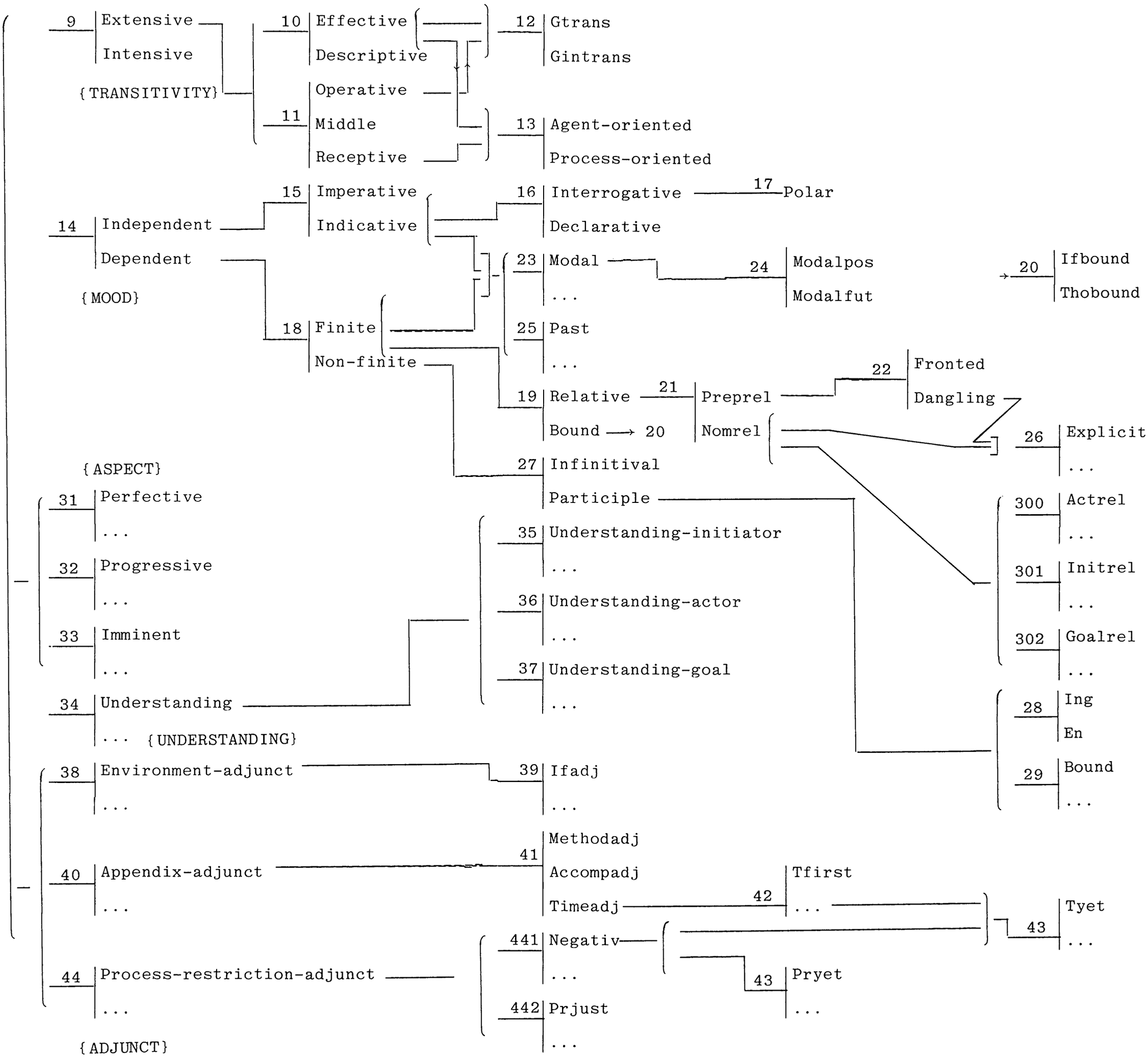
"The game hasn't ended".

It also provides the entry condition for system 443, which marks {Pryet} a clause with "yet" accompanying the verb:

"It hasn't yet ended".

The distinction between systems 43 and 443 is ad hoc; in system 43 {Tyet} marks a clause in which "yet" appears in appendix position:

"It hasn't ended yet"


9 Extensive
Intensive
{TRANSITIVITY}
10 Effective
Descriptive
11 Middle
Receptive
Operative
12 Gtrans
Gintrans
13 Agent-oriented
Process-oriented
14 Independent
Dependent
{MOOD}
15 Imperative
Indicative
16 Interrogative
Declarative
17 Polar
18 Finite
Non-finite
23 Modal
...
24 Modalpos
Modalfut
25 Past
...
19 Relative
Bound → 20
21 Preprel
Nomrel
22 Fronted
Dangling
20 Ifbound
Thobound
26 Explicit
...
27 Infinitival
Participle
{ASPECT}
31 Perfective
...
32 Progressive
...
33 Imminent
...
34 Understanding
... {UNDERSTANDING}
35 Understanding-initiator
...
36 Understanding-actor
...
37 Understanding-goal
...
300 Actrel
...
301 Initrel
...
302 Goalrel
...
28 Ing
En
29 Bound
...
38 Environment-adjunct
...
39 Ifadj
...
40 Appendix-adjunct
...
41 Methodadj
Accompadj
Timeadj
42 Tfirst
...
43 Tyet
...
44 Process-restriction-adjunct
...
{ADJUNCT}
441 Negativ
...
442 Prjust
...
43 Pryet
...

The motivation of the alternative positions of "yet" has not been explored.

System 40 marks a clause which has an adjunct in clause-final position. The appendix may convey the method used for the main-clause activity, {Methodadj} in system 41:

"I won by completing my edge"

or may convey an "accompaniment", {Accompadj}:

"The game began with a kick into touch".

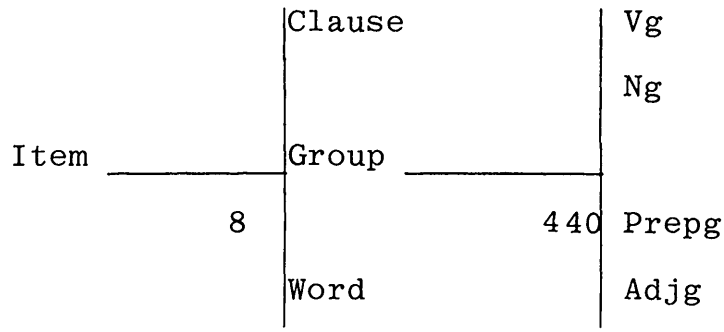
If the appendix gives a time-reference the clause is {Timeadj}, either{Tfirst}

"You saw it first"

or, as we saw, {Tyet}. Appendix-adjuncts were so called because in the program's output adjuncts in this position normally expand upon information already given. Yet "first" in the last example, though an appendix-adjunct, provides a time reference as we said an environment adjunct does. The {ADJUNCT} systems employ a purely positional criterion which is plainly inadequate.

4.5 Group Systems

There follow the systems networks for the groups which occur in the grammar. They are linked to the rest of the network as follows:



4.5.1 Verb-group systems

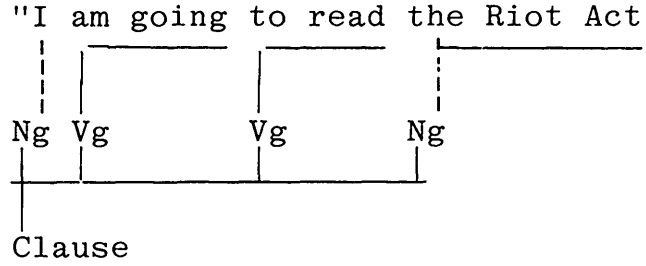
In the course of examining the clause systems we noted that all verb forms are taken to be immediate constituents of the clause. This statement is not to be taken as meeaning that every verb word is a separate immediate clause constituent, because sometimes more than one verb-word is needed to realise a particular function. For example, there are two forms of the infinitive, one of which has an explicit "to":

"(I'd like) to teach (the world to sing)".

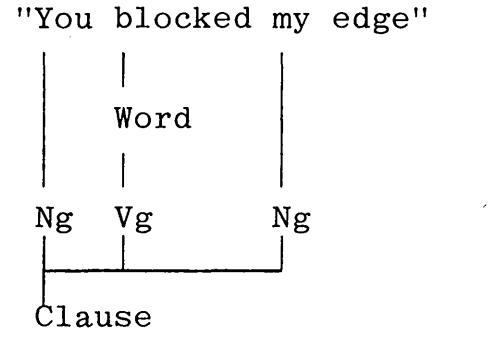
while the other doesn't:

"(You can't) teach (the world to sing)".

Both "teach" and "to teach" are {Vg ---} items. Similarly, the IMMINENT function is realised by "be going", as in the following:



The verb words of the clause are immediately dominated by the {Vg} items, and no {Word...} item intervenes. The reason is that the majority of Vg items have no constituent structure of {Word...} items, and the additional layer of structure is therefore superfluous:



It would probably be a preferable solution to delete {Vg} from the {TYPE} system, and to make all verb words immediate {Word} constituents of the clause. The existence of multi-word verb forms, such as the infinitive with 'to' motivated the present arrangement, but more fully considered should have led to another conclusion. We may notice in particular that in the case of 'be going', a variety of adverbs may intervene between finite forms of "be" and "going":

"You are always (never, soon, not,...&c) going to..."

The rules which say what order auxiliary verbs go in, which verb is finite, what happens when the clause is a question, and where adverbs go, are all clause rules,

set out in the next chapter. The verb-group rules contain no provision for dealing with these problems. The present system is therefore wrong, and should be revised by the excusion of {Vg} from the grammar, the addition of some few clause rules, and the constituent-structure assertion that each verb word is an immediate constituent of the clause.

System 45 distinguishes lexical, or content, verbs from grammatical, or auxiliary, verbs. The category of grammatical verbs is distinguished in system 46 into modal and non-modal, where modal verbs are "can, may, will, ---", distinguished in system 47. Non-grammatical verbs are either neutral, carrying only tense and number, as

"Did (you see Jane.")

or aspectual, carrying some aspect as well:

"Have (you seen Jane.)"

It is possible that "do" is a neutral carrier only when imported into questions as above, negatives as:

"I didn't (see a thing, officer)."

and negative environment inversions, as:

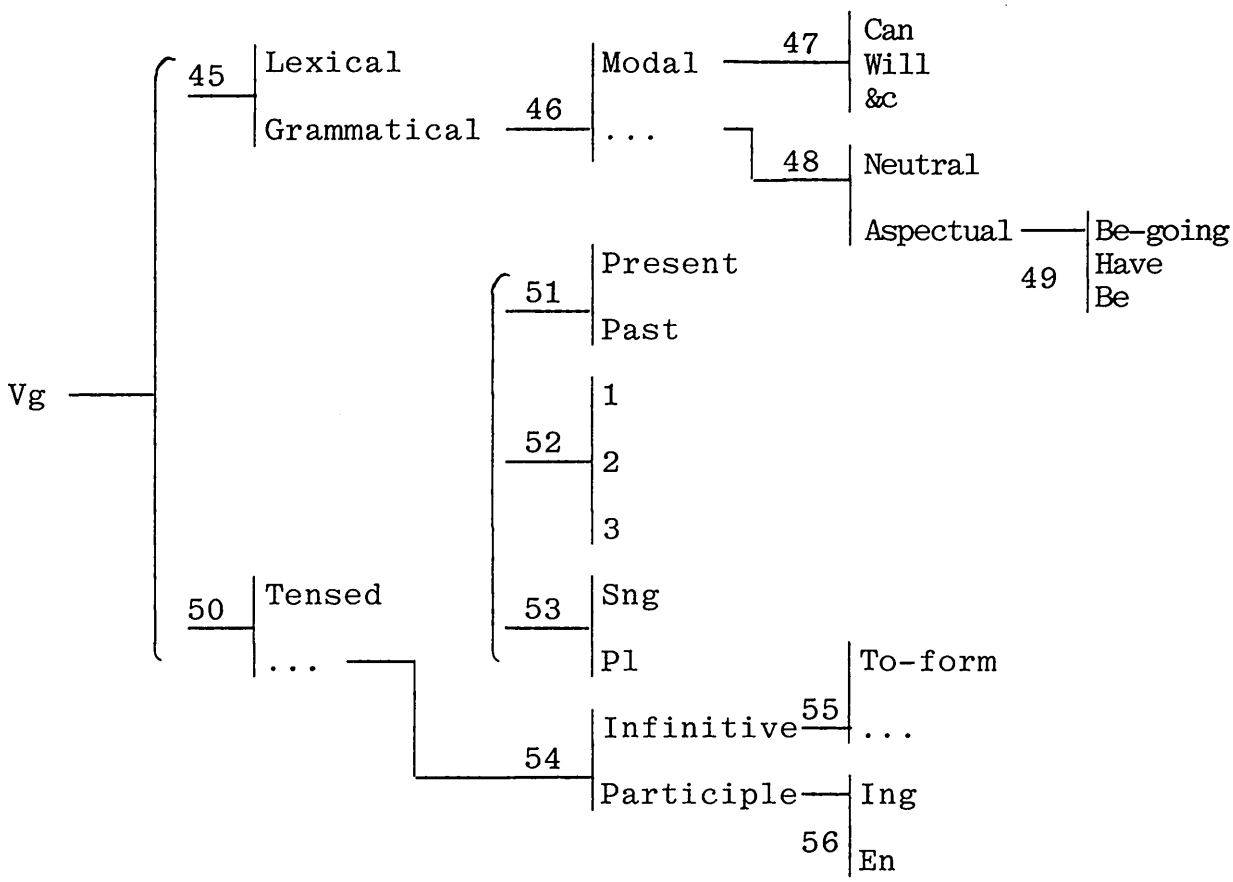
"Not only did (the voles eat the corms, they ate the flowers too)".

When "do" is emphatic:

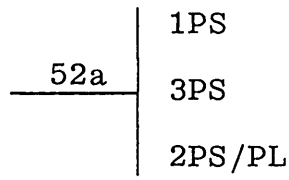
"I do wish you'd stop that)"

perhaps we should regard it as aspectual.

Verb-group systems network.



System 50 distinguishes tensed from non-tensed verbs. Only if the verb is tensed do we enquire whether it is present or past, and what person and number it is. Systems 52 and 53 are somewhat redundant. The second person singular, and the plural, have the same form in English, so an alternative system would be:



but the present system seems clearer. System 54 distinguishes infinitive from participle forms, and system 55 marks the 'to' form of the infinitive:

"She told me to come"

from

"She bade me come".

System 56 distinguishes present participle and gerundive forms from past participle forms:

"The planting of bulbs is the dull bit; once planted, they grow..."

4.5.2 Noun Group Systems

The Ng systems are designed to classify an item in such a way that we can tell how it is made up and what each constituent is doing. We are not here concerned with the role of the Ng in the item which includes it, except where the role has an effect upon the form of the Ng. For example, if a personal pronoun is the object

of a active transitive verb or a preposition we must mark the fact, because some of the pronouns have special forms, "me, him, her, us, them" in the accusative.

The first distinction made is in system 60, between those noun-groups which are clauses and those which are nominal. A {Clause} Ng is a clause marked {Dependent} in system 14 of the clause {MOOD} systems; it does the job of a noun or pronoun in the clause. Examples of {Clause} Ngs are:

"(We all know) that Fred is Australian"
whether Fred is Australian"
why Fred went to Australia"

and

"To be an Australian (is the height of bliss)".

However, we must distinguish {Clause} Ngs with " -ing" verbs, from the rather similar {Nominal} Ngs with the {Gerund} feature, system 70:

"Praising Australia (pleases the natives)"

is an example of a {Clause} Ng, whereas:

"The praising of Australia (is best left to Australians)"

is {Nominal Gerund ---} . We make the distinction by observing that a gerund is a noun and so can have an article in front, as it had in the last example. It can be modified by an adjective, and classified by a

noun:

"Constant camel riding (makes you seasick)".

It can't have a direct object, but instead uses a preposition-group qualifier:

"Constant riding of camels...".

Contrast a {Clause} Ng:

"Constantly riding camels...".

It will be clear that according to these criteria

"her dancing"

might be either a clause or a gerund Ng. As a clause Ng it would be equivalent in meaning to "The fact that...", while as a gerund it would mean something like "Her action...". So in:

"Her dancing on his grave I could scarcely credit"

the item is {Clause}, and in :

"Her dancing lasted an hour"

it is {Nominal}.

A problem arises in the treatment of the explicit subject of a participle-clause Ng. The subject goes in the possessive form:

"My (John's) singing Lieder wrecked the evening".

One's instinct is to say that the subject of the clause must be an immediate constituent of the clause. However we cannot treat it as a Ng for the purpose, for it behaves not as a Ng but as a determiner. The subject is a 'possessive adjective' if it corresponds to a

pronoun, as in the last example. We therefore treat the subject of a participial {Clause} Ng as an immediate constituent not of the clause but of the Ng, and as a possessive determiner. The other determiner options are not open to it and the network captures this fact in defining the entry conditions for system 73. We shall return to the determiner in due course, after first examining systems 61-64.

Systems 61 -64 classify the Ng as a whole. System 61 distinguishes singular from plural according to the number of things meant by the Ng. System 62 distinguishs first, second and third person Ngs, and system 63 further distinguishes third - person Ngs into animate and inanimate. Systems 62 and 63 are not integrated into the network in the most economical way, since we know, for example, that only {Proref} pronouns can be first -,or second, person participants in a process. However, they are set out as they are to correspond with the logic of the program, which benefits from the simplification of entry conditions to the systems at the expense of distorting somewhat the analysis of the options available. System 64 distinguishes definite from indefinite Ngs. Both "plural" and "definite" are assumed for the moment to be well defined features: in the chapter after next we explain exactly what makes an Ng plural or definite for the program's purposes.

The present grammar is designed to govern production of English: the reason for having system 64 classify Ngs as {Def} or {Indef} is that sometimes a special constituent has to be added to the Ng to convey this feature of it. In many cases no special constituent is needed; an independently motivated constituent conveys the feature. For example, somebody whose name is John can be denoted the simple Ng "John"; proper nouns of themselves convey the definiteness of the Ng in which they occur. (We shall mention exceptions later). Similarly, the quantifier "some" conveys indefiniteness, and "some people" is an indefinite Ng. On the other hand, a common noun by itself, such as "dog", needs to be preceded by the definite article if the resultant Ng is to be { Definite }, "the dog". Therefore system 64 is an abstract classification of the Ng, which takes no account of how the definitness, or otherwise, of the Ng is conveyed. The structure-building rules, given in the next chapter, determine what constituent of the Ng conveys the feature, adding one if necessary to do the job.

This method of dealing with the problem of definite and indefinite Ngs, and with the variety of ways by which the feature may be conveyed, has the advantage of simplicity. System 64 classifies the Ng as a whole, whereas alternative formulations have normally involved classifying for definiteness all the constituents which might convey this feature; this complicates the classifi-

cation and distracts attention from the fact that definiteness is an Ng feature corresponding to the meaning of the Ng as a whole, not that of a constituent only. The present systems could usefully be compared with Winograd's (1972 p58).

The program's output includes no quantifiers. The problems of quantified Ngs are notorious, and no attempt is made here to propose even tentative solutions. However, the entry condition for a {QUANTIFIER}system is marked, because it is suggested that quantification, like definiteness, should be analysed in detachment from a classification of the various particular constituents which may convey it:

"all the men, few men, a few men, a few, few, ---"

A preliminary sketch of some suitable structure-building rules encouraged the hope that this approach might work. It would replace Winograd's relegation of quantifier analysis to "the semantics" (1972 p 59).

Except for system 84 the remaining systems classify the Ng in terms of its constituents. Adapting from Winograd (1972: 56) we say that an Ng may have constituents:

Det - Ord - Card - Mod - Class - Nominal - Qual - GenMkr

Each is explained in due course. Each is optional in the sense that we can find an Ng without it, but the

order of whatever constituents are present in a particular Ng conforms to the order shown.

Systems 65-70 classify the nominal component in the list above. Usually it is the head of the Ng. System 65 distinguishes pronouns from non-pronouns, and system 66 proref pronouns from prostring pronouns because the options formalised in systems 69,71 -82 are not open to proref pronouns. A prostring pronoun is one which picks up an antecedent word string (compare chapter 3, section 3.2.1) whereas a proref pronoun picks up the referent of it. So "one" in

"the dead goldfish and the surviving one"

picks up the word "goldfish", not the referent of it. It is not entirely clear what prior word-string may constitute a prostring antecedent. It seems necessary that the word-string should be dominated by some constituent-structure node which dominates all and only all the word-string. An example suggested by Ritchie (p.c.) illustrates this:

* "He belongs to an old man's club but I belong to a young one."

This condition is probably necessary, but is evidently not sufficient, (Isard, p.c.):

* "It's actually a tea pot, but we use it as a coffee one"

These problems are left unresolved in the present program.

The possessive pronouns seem to be prostring pronouns in the same way, so that "yours" is equivalent to "your one", or "your ones".

"I'll light my rocket and you do yours".

This equivalence has two implications. First the possessive determiner "your" conveys definiteness, so "yours" must be a {Definite} Ng. Second, we should not expect to be able to add to "yours" any Ng component which in the list above comes between the determiner and the nominal components: if the Ng demands such a component we must write "your --- one(s)", as

* "three yours"

  "your three (ones)"

* "red yours"

  "your red ones"

However, a post-nominal qualifier is acceptable:

"yours to hand of the 13th inst."

and we shall see later that the Genitive-Marker may occur too.

All other pronouns are prorefs, so in:

"Gardeners dislike voles but owls love them"

"them" picks up the referent of the Ng "voles".

The program's output includes noun-groups from which the prostring "one" may be left out: both

"the other one" and "the other"

are acceptable, and they seem to be equivalent in

meaning. The latter of these two is taken to be an Ng whose head is an adjective. An alternative would be to take "other" as a prostring pronoun, but there seems no good reason to do so when we remember that indefinitely many other adjectives may appear as the head of an Ng. The construction in

"the unjust has the just's umbrella"

as R A Knox said of the umbrella thief, is quite acceptable. A second alternative would be to say that an adjective at the head of an Ng ipso facto becomes a noun. In favour of this we might mention that diachronically certain adjectives undoubted become nouns, as

"(the Dean was) a Red"

and that these may take the plural termination "-s":

"(Don't use detergent on) your woollens"

"(some Deans are) Reds"

But other adjectives do not take this form:

"the rich (grow richer every day)"

and there are many whose behaviour would differ between idiolects. However, in the present grammar neither alternative has been adopted. Instead it has been assumed that an adjective may be the head of an Ng, and that in such cases some adjectives idiomatically take the plural termination "-s".

Systems 67 and 68 mark deictic pronouns, and distinguish proximate ("this, these ---") from remote

("that, those ---"). If not deictic a pronoun is unmarked, in system 67. If unmarked, a pronoun usually has a special form ("him,---") when the object of an active transitive verb or of a preposition. For this reason system 84 distinguishes subject and object. It is obvious that the placement of system 84 means that this distinction is made for every Ng, regardless whether the form of it is affected by being {Object} or not. The network correctly reflects the program logic which was simplified by this redundancy, but in fact system 84 really has to be entered only in quite a small number of cases since in the main English does not realise the SUBJECT and OBJECT functions in features of the Ng.

The systems which classify an Ng in terms of constituents other than the nominal, namely 71 - 82, are not open to proref pronouns. The reason is that all these constituents focus or restrict the nominal until the whole assemblage refers to its intended referent and no more. But, as we noticed in chapter1, a proref pronoun has deictic force and picks up the referent of its antecedent. So all the focussing has been done already and the proref needs, indeeds permits, none. That is also the reason why proref pronouns are definite: a definition of definiteness is given in the chapter after next, but we can state informally that an Ng is definite if the number of entities we intend to refer to coincides

with the number of entities referred to by the Ng we construct. Since a proref pronoun refers to exactly those entities denoted by the antecedent, it must be definite.

System 70 completes the specification of the Nominal component. It distinguishes common nouns, {Noun} names {Propernoun}, and gerunds{Gerund} , which are distinguished in the way we mentioned in describing system 60. It is arguable that the grammar should not treat propernouns and common nouns as though they could occur in much the same constructions: for example, we can't normally put a determiner in front of a name:

* "The Richard is my nephew"

because "Richard" is definite without the addition of "the". But this rule is not applied in certain idioms:

"The McEachans (have an osprey in their garden)"

nor when the propernoun is preceded by a possessive determiner:

"Your Jane punched my Jimmy"

In the latter example "Jane" without a determiner would be a definite Ng, but the possessive has the effect of making "Jane" the name of a class of children called Jane and simultaneously selecting the one particular child in question. A proper name can be made generic:

"A Julius Caesar (is what you want for throwing bridges over rivers)"

and may be the name of a class:

"An Argyllshire Campbell is anathema to a MacDonald"

There is also the idiom whereby a man's name is taken to refer not to the man, but to the set of men the man has been, might have been, and so on. So we get:

"The Professor Smith (I admired bore no relation to the public's impression)"

"The now Sir Geoffrey Jackson ---"

In the program's output names are used only in the singular, and always realise the definiteness function, so a determiner is never needed. A more comprehensive grammar would somewhat elaborate the rules, but would leave names in parallel with common-nouns as they are here

To return now to system 71, {Determined} marks the presence of a determining constituent motivated independently of definiteness, system 64. So deictic adjectives ("this, those, yonder, ---"), possessive adjectives ("his, their ---"). and possessive noun-groups ("Bill's the fat fellow's,---") are determiners, but articles ("a, the"), which are inserted simply to convey definiteness or indefiniteness when nothing else will, are not. The next chapter, section 5.7, shows how the rules which derive constituent-structure from a feature-list insert articles when needed.

The determiner of a nominal Ng may be deictic,

system 72, and if so must be proximate or remote, system 68:

"this chicken" or "that egg".

If possessive, the determiner is a possessive adjective,

"your horse"

or a genitive noun-group

"the new computer's arrival".

Systems 74 -82 classify the Ng with regard to its remaining components. System 74 marks {Ordered} Ngs, those modified by an ordinal adjective such as "first, second --- last" and "next":

"the last dance".

System 75 marks Ngs with a number word, whether the number word comes first:

"five voles"

or after a determiner and perhaps an ordinal:

"the first five voles"

or at the head:

"(I'll have) five", "(I'll have) five of those".

System 76 marks an Ng modified by any Adjg not already distinguished:

"the Gordian knot, smoky glass, the largely denuded slopes".

Adjective groups are explained more fully in section 5.4 of this chapter.

An Ng is {Classified}, system 77, if one or more nouns precede the head noun and modify its meaning. In:

"a lawn mower"

"lawn" is a classifier. A classifier may be complex, having a modifier:

"a wild goose chase"

a number word:

"a five pound note"

a classifier, as "lawn" classifies "mower" in

"the lawn mower handle"

or a qualifier of a simple sort:

"your tax on off-street parking proposal".

But it cannot have an article or determiner of any kind, so although:

"a London fog"

is perfectly acceptable, and "the City" is a quite usual alternative for "London",

* "a the City fog"

is unacceptable, even if the oddity of the two adjacent articles is reduced by inserting some words:

* "a thick, choking, the City fog".

This is the most important respect in which classifiers fall short of being ordinary noun-groups. We need attach less significance to the observation that the meaning of a classifier may vary with the headnoun it precedes, and that the meaning of a classifier may be more restricted than that of the same word as a headnoun. For example in:

"they hitched along the Bath road"

"Bath" is at once understood to denote the Somerset city, whereas in:

"they married in the Bath chapel"

the word would as immediately be interpreted as a reference to the order of chivalry. Or notice that "pile" in "a pile driver" can denote only the support of a structure, whereas "pile" as head noun has a much wider range of possible meanings. These facts have been felt to discourage the view that a classifier is a noun group like any other, but we may remember that many noun-groups take on a meaning determined by context and in particular by the main verb. In:

"The athletes lost speed"

and "The athletes shot speed"

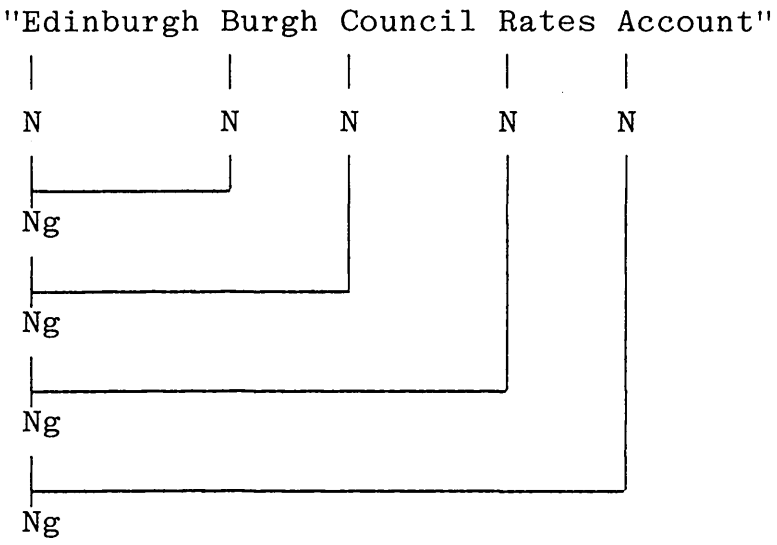
"speed" means respectively velocity and amphetamine drugs. We may conclude for the moment that a classifier is a noun-group subject to the limitation that it cannot have a determiner constituent.

We should now mention the fact that the classifier of a headnoun may be not a single noun-group but several:

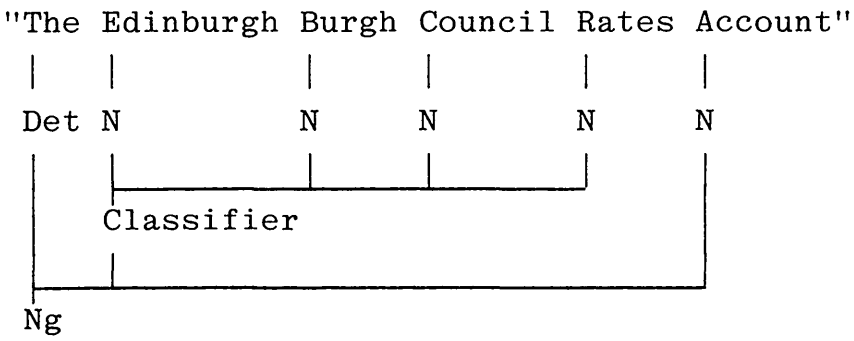
"Edinburgh Burgh Council Rates Account"

The headnoun is the heart of the semantic onion: the "Account" is classified as a "Rates Account", a "Council Rates Account" and so on successively as constraints are added. What the constituent structure of this group should be is not obvious. A left-branching structure

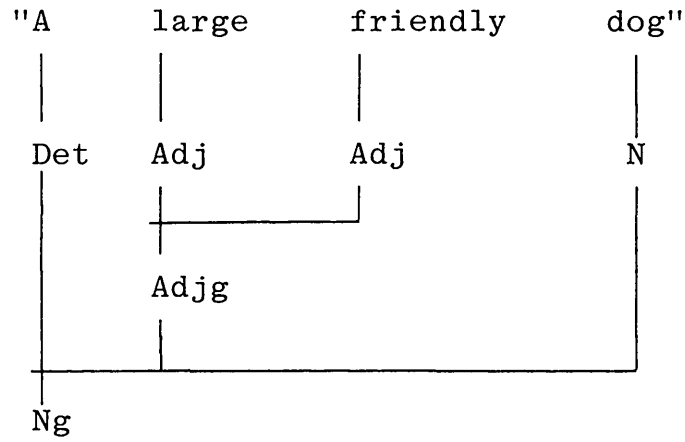
might seem to correspond best with the construction of the onion:



However, such structures are unattractive in psycholinguistic theory because of the memory load which they are considered to impose in recursive decoding. Furthermore, we might prefer to emphasise the resemblance between a string of adjectives, "a large friendly dog", and a set of classifiers, by assigning them similar structures. So we assume for the moment that the structure of the last example is:



comparable with:



Adjective groups are considered later. The assumption made about classifiers is that a classifier is a string of one or more noun-groups of which none may be determined. The program in fact produces no classifier, but were it to do so, the rules would define a classifier as a noun-group with a {Classifier} feature; this feature would impose the appropriate structural constraints upon the string.

After the head there may occur a qualifier or qualifiers, system 78. The qualifier is an adjective group:

"Nature red in tooth and claw".

a preposition-group:

"the worm in the rose",

or a clause:

"the electricity that he stole from the G.P.O."

"the Guinness he lavished on his tomatoes".

There may be more than one qualifier, in which case the preferred order of qualifiers is Adjg - Prepg - Clause, and this rule is one of the structure-building rules in the next chapter. However, the order may be affected by other factors which the present grammar ignores. A particularly long qualifier is likely to be postponed:

"the square which is empty, adjacent to the corner which you have just taken"

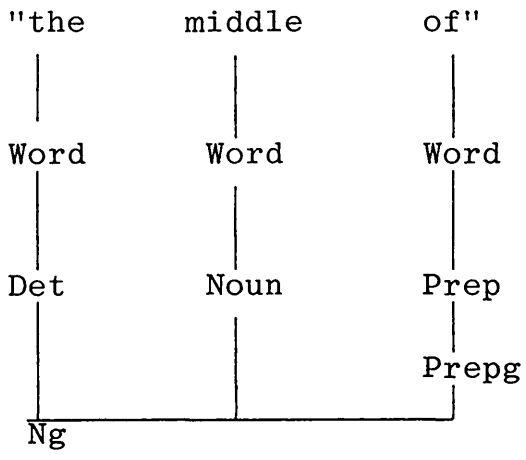
and others have suggested that a restrictive qualifier always more closely follows the head than a non-restrictive one does:

"the shoes he wore, caked with mud, ---".

System 81 marks preposition-group qualifiers with dangling prepositions, when the prepositional object has been relativised and moved to the front of the clause (system 22÷):

"(the funeral he laughed in) the middle of".

It is assumed that in constituent structure the dangling preposition occupies the same position as it would were the prepositional object following:



The grammar does not handle examples where the {Dangling} is at a lower level. Usually such examples sound clumsy :

"the tree she made that pie of apples off"

although exceptional idioms are acceptable:

"the cup with the wine in"

To account for these would require an extension not of the systems network but of the structure-building rules in the next chapter, so that the dangling preposition marker could be passed "down" to the requisite depth.

System 83 marks the case in which the whole Ng is genitive being followed by "'s" or "'", as

"Bill's (boat)"

"cows' (milk"

"that one's (height)"

and even

"the cow that jumped over the moon's (flight authorisation)".

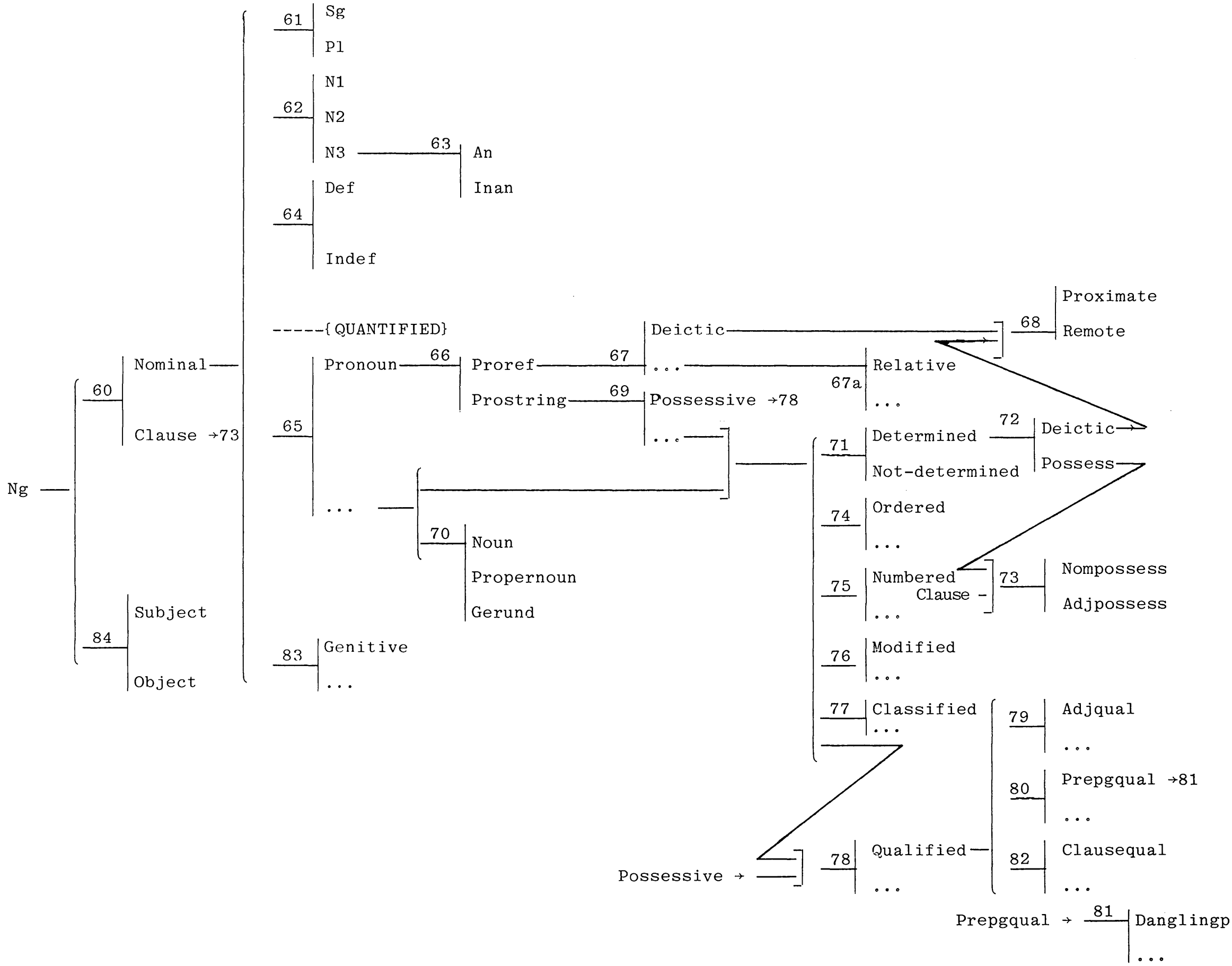
We noticed earlier that possessive pronouns can occur in {Genitive} Ngs:

(A. "How do you recognise your goldfish?")

B. "Mine's (colour is darker,)"

although they sound clumsy in the other cases ("His' is father") for obvious reasons. Normally the head of a {Genitive} Ng cannot be a deictic or unmarked pronoun, and

NOUN-GROUP SYSTEMS



"This car is faster but that's engine is quieter." sounds odd. However the dialect of Edinburgh permits "that" to be used of children or annoying adults, and so one hears:

"That knocks that's pipe out over my carpet."

4.5.3 Preposition group systems

The preposition-group is a simple structure, comprising a preposition followed by a noun-group. If the Prepg is {Dangling} the noun-group is not explicit at that point, but is understood from somewhere else (systems 22,81), where it may be explicit:

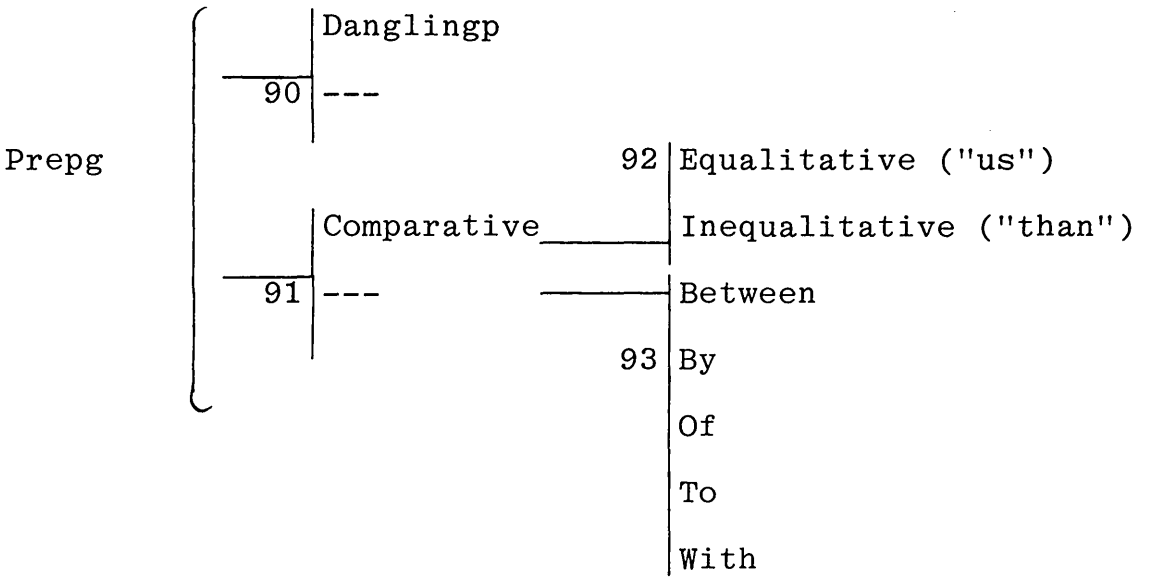
"---(the fellow) whom you borrowed the car from"

or implicit:

"---(the fellow) you borrowed the car from".

The present grammar makes no provision for the preposition word to be multiplex, such as "in between" or "up to": unlike Winograd (1972 p 60) we do not regard "in the middle of" as a complex preposition because its structure is that of a noun group and the semantic specialists in the program assemble it as such. However, a more complete grammar would certainly have to cope with multiplex prepositions, such as the two mentioned a moment ago.

The systems are



It will be clear that the sole purpose of systems 91,92 and 93 is to specify the preposition required. It is debatable whether the selection of a preposition is a grammatical matter, or whether it should more properly be left to lexical procedures. The decision to make the preposition a grammatical feature can be justified on the grounds that prepositions are drawn from the closed-class dictionary, and that they have some grammatical rather than lexical correlates, in particular case and number. In English, of course, all prepositions take the accusative, since we regard:

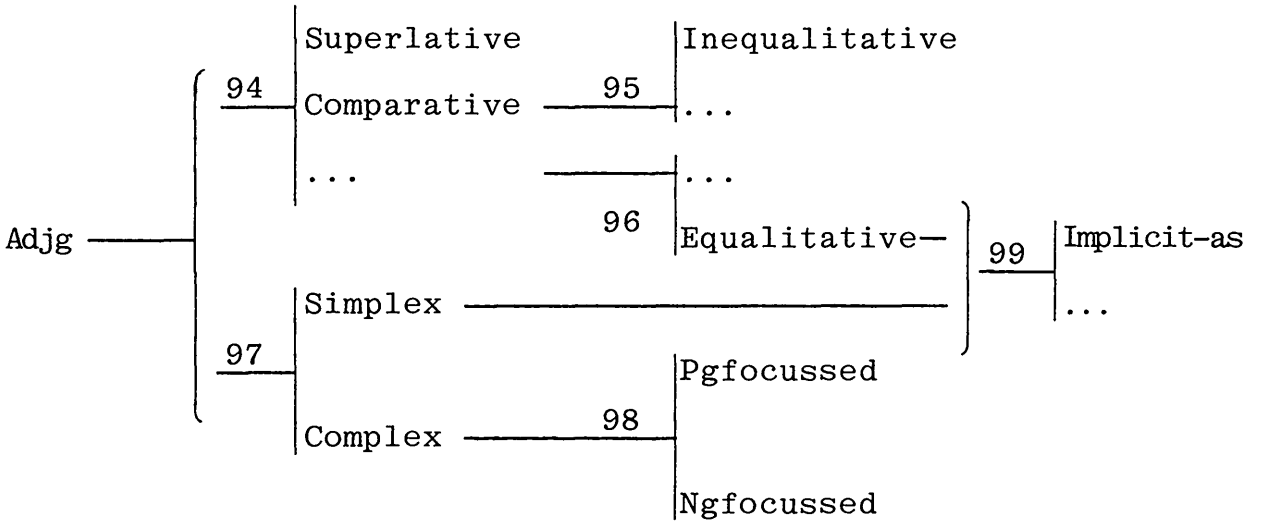
"(He's bigger) than I"

as either abbreviated for "---than I am", or simply wrong. The same is true of "us". However, in languages with more than one oblique case prepositions differ in this respect. So far as number is concerned, some

prepositions, such as "between", dominate only {Plural} nominals, and we might regard this as a grammatical constraint. In the present program it happened to be easy to specify the required preposition by means of a feature of the Prepg, but relatively difficult to leave the decision to the constructor function of the preposition word, and so the former course was adopted.

4.5.4 Adjective-Group

The Adjg network is:



System 94 distinguishes the form of the adjective into the conventional Superlative ('largest, most populous'), Comparative ('larger, more populous'), and unmarked. Any of these may be {Simplex}, system 97, in which case it occurs alone, or {Complex}, in which case it is followed by a focussing phrase. A focussing phrase is a preposition group, {Pgfocussed}, as in:

"stately in her carriage, slower on the uptake, fastest on the draw"

except a preposition group introduced by "than" or "as" which we specially distinguish in systems 95 and 96; otherwise it is a noun-group, {Ng focussed}, as:

"opposite the hotel, more like her mother,---"

There are, of course, other ways an adjective may be focussed, for example by an epexegetic infinitive:

"ready to burst"

but we ignore these. It might be objected that "opposite" and "like" in the {Ngfocussed} examples are not adjectives at all, but are prepositions. The reason for taking them as adjectives is that the constructor procedures select "opposite", for example, before deciding whether the context requires that the focussing phrase be explicit, or not. If the former the Adjg will be {Complex} and the rules will put it after the noun:

"--- the one opposite the corner you had just taken"

otherwise it is {Simplex} :

"--- the opposite one".

There seems to be no good reason for saying that "opposite" is an adjective in the{Simplex} case but becomes a preposition in what we have called the {Complex} case, particularly when its semantic stable-mate "adjacent" remains an adjective in either context, "the adjacent edge" or "the edge adjacent to that".

A "than" phrase may occur only after comparative,system 95, and may be doubled up with a focussing phrase:

"slower on the uptake than his friends"

when it generally follows the focussing phrase. In system 96 "as---" phrases are distinguished, since they can appear only in the unmarked case in system 94. These too can have a focussing phrase but in that case must have an "as" preceding the adjective:

"--- as sound in the head as you are"

This "as" can be left out, {Implicit -as} in system 99, if the Adjg is {Simplex} :

"--- sound as a bell"

We observe that an Adjg may realise one of three functions. In an Ng it may be a MODIFIER or QUALIFIER, realising the features of systems 76 or 78-79. In a clause it may only be an ATTRIBUTE, system 9. There are constraints upon the form of an Adjg in realising some functions: in producing English the constraints are specified via the realisation rules of the functions concerned. We shall see shortly that, for example, Adjgs which realise the QUALIFIER function must be {Complex}. A {Simplex} Adjg may realise a MODIFIER:

"a cheery soul"

an ATTRIBUTE:

"(he looks) miserable"

but not a QUALIFIER except in legal jargon or verse:

"the minister plenipotentiary, the valley shady".

So the rules in the next chapter ensure that the QUALIFIER function is done by a {Complex} Adjg. {Simplex Comparatives}, with {--- Superlative} examples in brackets, are:

"Our cleverer (cleverest) lads study Virgil" MODIFIER

"The valley sheep are fatter (fattest)" ATTRIBUTE.

A{Complex} Adjg may realise a QUALIFIER

"Nature red in tooth and claw"

or an ATTRIBUTE

"He's certain of a win in the 4.30"

but generally not a MODIFIER. {Equalitative} and {Inequalitative} Adjgs are unusual, but do occur, with this function:

"our whiter than white teeth".

Just as there are comparative and superlative Adjgs which are {Simplex} so too there are comparative and superlative Adjgs which have a focussing phrase but no "than---" or "as---" phrase:

"the valley sheep are longer in the staple".

The systems set out here permit all of these combinations, and are thus less constrained than the alternative of Winograd (1972 p 61) for example. The stylistic constraints which are felt to be necessary, for example the banning of {Complex} Adjgs realising a MODIFIER, may be imposed in the structure-building rules.

Adjective strings occur, either as lists:

"An active old ghillie"

or divided by commas:

"A slow, methodical worker"

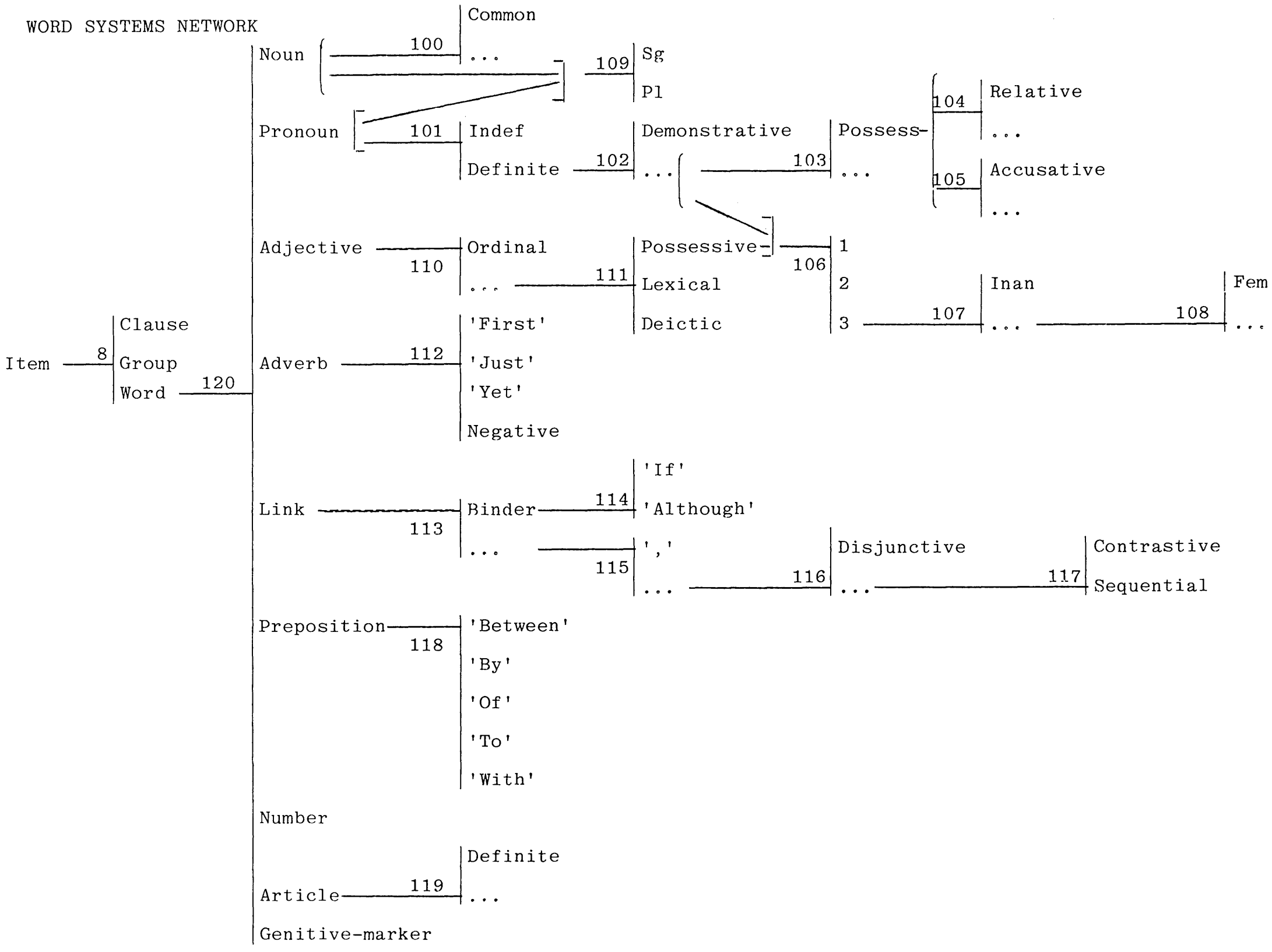
or by a conjunction:

"An entertaining but unfounded scandal".

All such Adjgs are {Coordinate}. An examination of the {COORDINATION} systems, section 3, show how such strings are characterised. Coordinate Adjgs often realise a MODIFIER, as in the three examples above, but are less common as realising a QUALIFIER, or ATTRIBUTE. The grammar incorporates no rules to govern this kind of example because the rules would be stylistic and idiolectal; when the program needs to produce a co-ordinate Adjg it assumes that it may do so wherever the corresponding simple Adjg would be permitted.

4.6 Word Systems

The {WORD} systems offer simple categorisation of the words used by the system. We remark just two points. First, certain features appear in quotation marks. This indicates that the feature-name actually is the word in question. In such cases the program procedure which determines the word does so by picking it from the word-constituent feature list. In other cases it is more convenient to supply the procedure with features of the sort appearing in systems 101-108. Secondly, there is no {Word Verb---} option. As we saw in the section describing Vgs, the {Word Verb---} options have been absorbed into the {Group


WORD SYSTEMS NETWORK
Item
8
Clause
Group
Word
120
Noun
100
Common
...
109
Sg
Pl
Pronoun
101
Indef
Definite
102
Demonstrative
...
103
...
Possess-
104
Relative
...
105
Accusative
...
Adjective
110
Ordinal
...
111
Possessive
106
1
2
3
Lexical
Deictic
107
Inan
...
108
Fem
...
Adverb
112
'First'
'Just'
'Yet'
Negative
Link
113
Binder
114
'If'
'Although'
...
115
','
...
116
Disjunctive
...
117
Contrastive
Sequential
Preposition
118
'Between'
'By'
'Of'
'To'
'With'
Number
Article
119
Definite
...
Genitive-marker

Verb-Group ---} options. The grammar thus contains no items which are verb-words. The reason has been mentioned already.

System 100 distinguishes common-nouns, conventionally spelt with a miniscule initial in English, from proper-nouns. It ignores the problems of generic uses of proper names, definite descriptions, and other such issues. System 101 marks as {Indef} the prostring pronoun "one", systems 102 and 103 mark demonstrative and possessive pronouns in the obvious way, and system 104 distinguishes relative pronouns from those otherwise unmarked ("he, she, we,---"). Only in the case of relative or unmarked pronouns is it necessary to mark the accusative form, system 105. Systems 106 -108 apply to all pronouns except indefinite and deictic ones, and to possessive adjectives; they serve to distinguish person ("mine, yours"), animacy ("hers, its"), and gender ("his, hers"). Number was dealt with in system 110. The remaining systems are designed to be self-explanatory.

4.7 {COORDINATION} systems

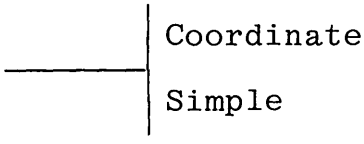
Chomsky (1957) suggested that a good test whether a string 'X' and a string 'Y' were phrase structure constituents and were of the same type was to replace 'X' by 'X and Y' and see whether the item in which 'X' originally occurred remained well-formed. Winograd's

AND demon (1972 pp 22,60) likewise attempted to find a pair of syntactically equivalent items for the coordinate conjunction to link. Chomsky's suggestion, of course, is not the whole picture, since we can quite well coordinate items which are not syntactically equivalent:

"He is a mathematician and very clever"

"He is a fool and up to his ears in debt"

are both well-formed, but that does not at all entail that, for example, the Adjg "very clever" behaves like the noun-group "a mathematician" in other contexts. In the present grammar the rule is put the other way round. Only if items are equivalent may they be co-ordinate, but "equivalent" is defined functionally not in terms of grammatical type. That is, coordinated items must realise the same bundle of functions within their particular environment. The grammar does not include this rule explicitly, but it is a consequence of the construction procedures. Each procedure must complete the feature list for its item by selecting the paradeigmatic features from the relevant network, but the first selection made is always from



If the item turns out to be {Coordinate---}, the procedure at once constructs the required number of descendants, copying over to each the feature list of the

{Coordinate} parent, but replacing {Coordinate} by {Simple}. The result is that all coordinated items are functionally equivalent, but, since their paradeigmatic features may differ, need not be of a common grammatical type.

The reformulation of the rule has advantages. It explains the facts, and in particular the fact that type-equivalence is a normal, but not invariable, concomitant of coordination. It explains the facts by attending to what coordination is for: the two examples exhibited a coordinate realisation of the ATTRIBUTE function because there were two things to say about the attribuant.
We can't say:

* "the and the old horse"

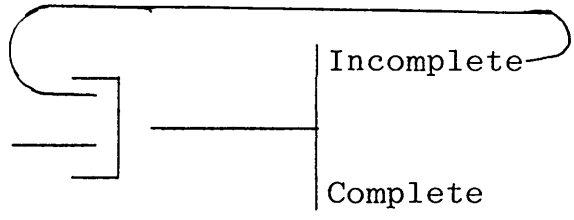
because, as we saw in section 5.2 of this chapter, the definiteness of the Ng is conveyed by just one article: there never could be any need for two. In contrast, other determiners which not only realise definiteness but convey something else too may be coordinated:

"Her and my bank account".

Before setting out the{COORDINATION} systems, we must say a word about a fundamental difficulty for a systemic grammar in producing coordinate items. Such items may contain indefinitely many conjoined constituents: the Homeric ship-list starts with 28 conjoined entries (Homer, 1902, Bk 2), and the English translation is grammatical, if dull. But a systemic grammar produces

by classifying. To produce a coordinated set, the grammar must classify a set of indefinite size, an enterprise which must fail.

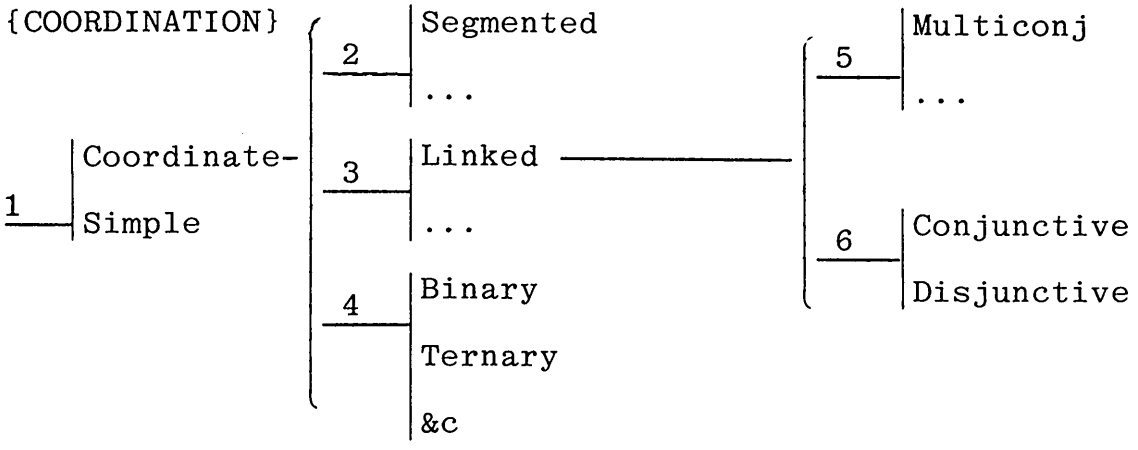
Hudson (1972a) proposed a solution which extended the grammar by the addition of a novel type of system. This system did not classify in the way that all the other systems do. Instead it constituted a counter, which added {Incomplete} to the item's feature list once for every conjoined constituent except the last, and {Complete} for the last, thus:



where {Incomplete} is itself an entry condition for the system. Although {Incomplete} is called a feature, it is plainly not a feature in the same sense as {Coordinate} or {Plural} or any other feature we shall meet. It is really a cuckoo's egg, camouflaged as a feature to secure the correct treatment from the Realisation and Building rules, which in due course hatch a conjoined item for each occurrence of {Incomplete} and {Complete}.

This proposal has the merit of coping with sets of undefined size, but it does so in a way which we shall not follow here. It is clear that systemic functional grammar cannot produce coordinate items without some extension

of its purely classificatory apparatus, but we do not feel confident that the right extension has been found. Meanwhile we adopt an admittedly inadequate solution, whose sole merit is that it really is a system of the same type as the rest. The {COORDINATION} systems are:



System 1 distinguishes coordinate from simple items. If the item is coordinate, three systems are entered simultaneously. System 2 marks the item whose coordinated constituents are separated by commas:

"I came, I saw, I conquered"

"She arrived, and he came with her"

from those whose constituents are not thus segmented:

"He walked up and down"

"A large old house".

System 3 marks the item whose constituent structure contains a conjunction:

"Wine, women, and song"

from those without, such as the string "Bread, kippers, tea, a fine meal".

If there is some conjunction, system 5 tells us whether every adjacent pair of the conjoined constituents is explicitly linked:

"(He's) a fool or a knave or both"

or whether only one pair is, the last:

"(He's) a knave, a fool, or both".

It is generally true of English that a coordinated set of items which has just one conjunction contains the conjunction in the final pair. However colloquial exceptions are frequent, as:

"Flour, salt and pepper, beans, rice"

but we ignore them. System 6 distinguishes disjunctive from conjunctive sets. We make the distinction because the two kinds of coordinate item differ in syntagmatic dependency. The features of the item, such as number, are given by just one member of the set:

"Jane or Jean is her name"

whereas a conjunctive set comprises all its members together:

"Jane and Jean are her names".

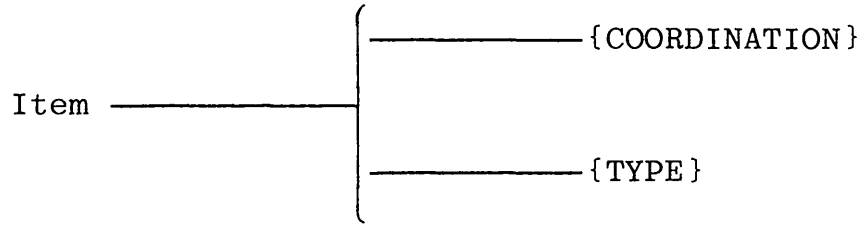
We do not further distinguish "and" from "but" because they are syntactically equivalent; in any case, we have no mechanism for getting the right order of "and" and "but" in a set of indefinite size:

"He came in, and I saw him, but she didn't notice"

"He came in, but she didn't notice, and I said nothing".

System 4 is our present system for determining how many items there are in the conjoined set. We have already disclaimed any generality for this system, and simply remark here that if there are two items in the set it is {Binary}, else {Ternary} if there are three, and so on.

As we saw at the beginning of this section, the construction procedures enter system 1 after the syntagmatic features have been derived from the function-bundles but before investigation of the remaining features. To represent this correctly would require a proliferation of arcs worthy of Guy Fawkes night, and we therefore represent the entry condition more simply, but incorrectly, as:



Because coordination is dealt with in the way described, by the procedures of the program, the normal construction cycle of features realised as functions realised as features is not used. There are no functions to realise features of the {COORDINATION} systems. In this respect the COORDINATION systems are unique, and will probably remain so after the theory has developed sufficiently

to account satisfactorily for the phenomena.

CHAPTER 5

FUNCTIONS, AND THE REALISATION AND BUILDING RULES

1 Introduction

This chapter explains how the features occurring in the systems in the last chapter are used. It sets out all the rules used by the program for deriving the immediate constituent structure of an item from its feature list, and it gives several worked examples. In this chapter and henceforth the word "function" will be used to refer only to grammatical functions: sections of computer code, which the particular computer language used for the program happens to call "functions", will be referred to by the more general term "procedures". We shall thus distinguish grammatical functions from computational procedures; the names of functions will always be written in capitals, as SUBJECT, distinguished from features, as (Plural), and names of subnetworks of systems, as (TRANSITIVITY).

The rules set out in this chapter are a transcript of part of the program. They therefore make clear exactly what the program does, but are correspondingly liable to criticism for various inadequacies and ad hoc devices. Some of these shortcomings are mentioned in the explanations of each set of rules, and more general points are collected at the end of the chapter. We say nothing about the interpreter procedure which operates the rules, except that it is extremely simple and would interpret any set of rules obeying the formation constraints of the

present set.

5.2 Functions

We are accustomed to the distinction between form and function, and to the idea that at least some grammatical items serve functions in their context. We accept, too, that a job being done by an item may influence its form. For example, it seems sensible to say that the noun-group "we" acting as SUBJECT of a finite clause has a different form from "us" acting as OBJECT of a transitive active verb just because of its differing function. Systemic functional grammar, however, goes further. It asserts that every grammatical item has one or more functions, and gives the term "function" a technical sense.

A function represents a grammatical constraint of the environment upon the form of the corresponding constituent item. We shall call the set of functions which an item has its "function-bundle", and the function-bundle represents all relevant syntagmatic constraints upon the corresponding item. It will be immediately obvious that if the function-bundle captures every relevant constraint there need to be many functions. The number of functions required expands rapidly with the scope of the grammar. Although the present grammar is simple and requires only a manageable number of functions, a more complete grammar shows signs of obesity and consequent ill-health. Furthermore, as the number increases, we find that some

functions do not have any intuitive plausibility as "jobs being done" by the corresponding item, but must instead be taken simply as capturing a syntagmatic constraint. These problems are considered at the end of the chapter.

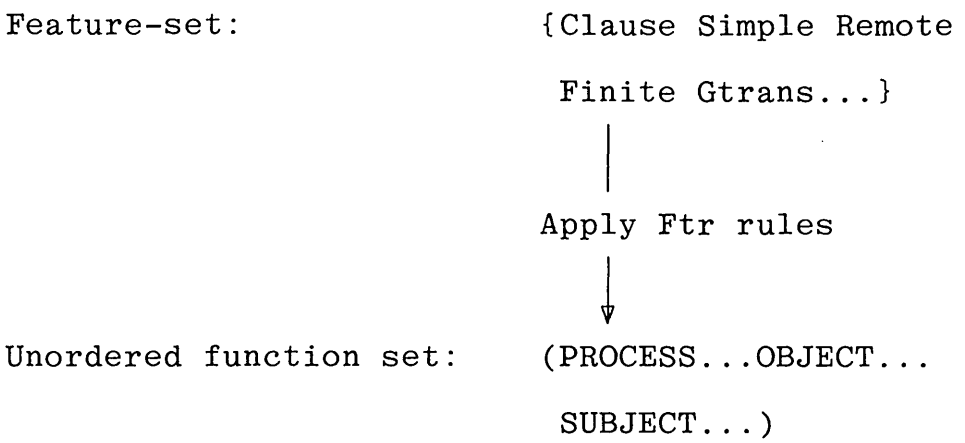
The grammatical environment whose constraints are expressed in the function notation is represented by the completed feature-list of the parent item. The feature-list contains all the information needed to determine immediate constituent structure, which is first specified in terms of an ordered set of function-bundles. We saw, for example, in section 1.2 of the last chapter that the clause feature-list (Clause Remote Finite Gtrans...) constitutes the environment from which the grammar enables us to derive the ordered function-bundle set

((SUBJECT...) (PROCESS...) (OBJECT...))

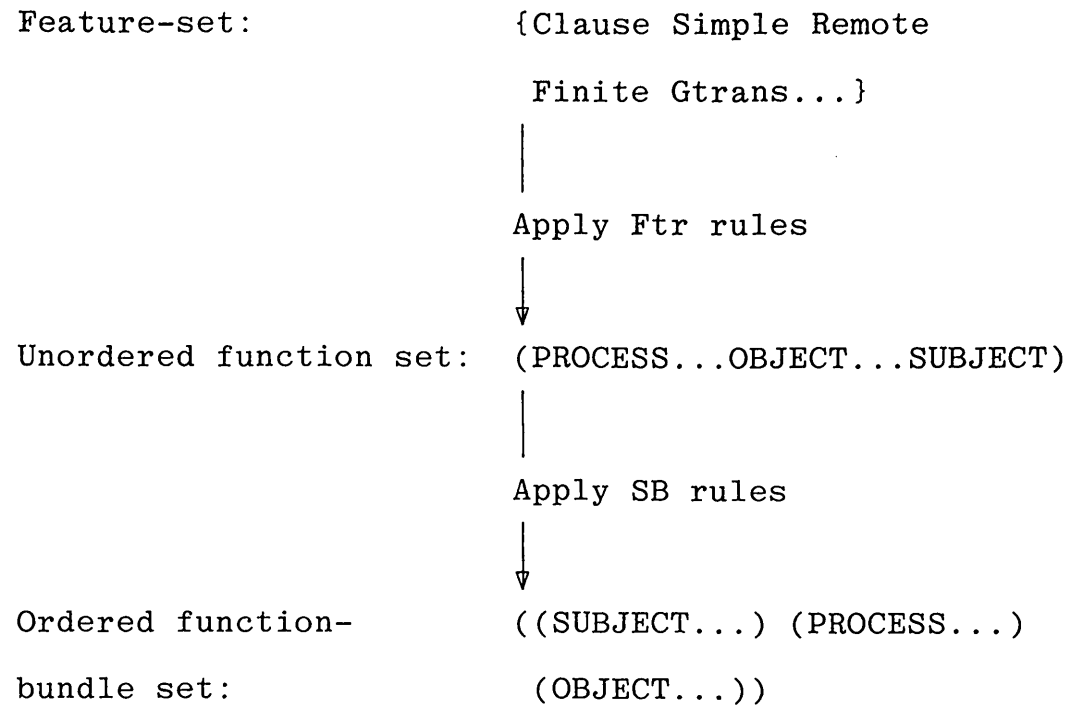
in which each bundle corresponds to an immediate constituent.

The ordered function-bundle set is derived from the item's feature-set by a two-stage process. First we apply rules which state what function must be performed by some as yet unidentified constituent of the item, given that the item has a particular feature. This relation of function to feature is a "realisation" relation: we shall say that features are realised in functions, and the rules which govern feature-realisation are "feature-realisation" or Ftr rules. For example, a clause Ftr rule states that

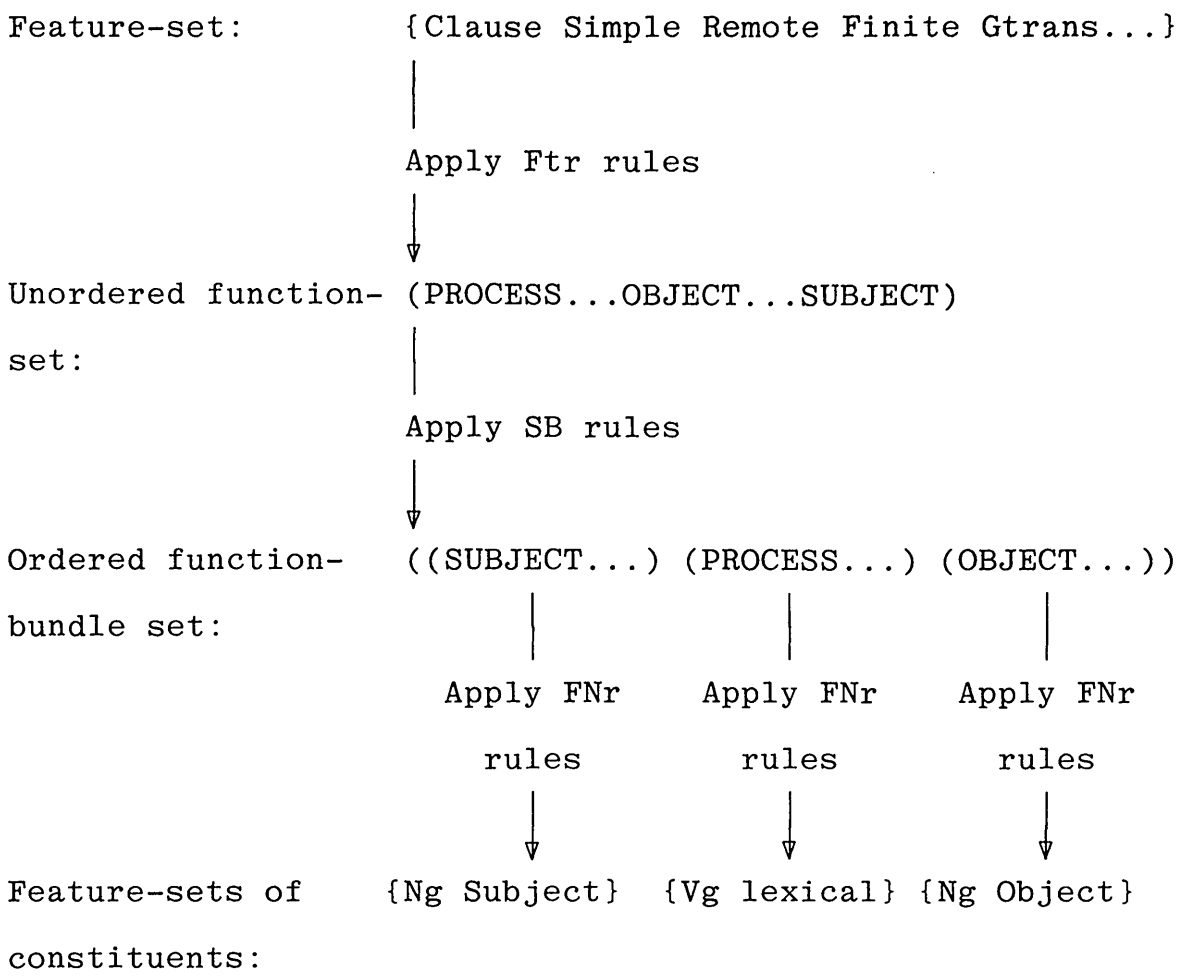
the {Clause} item, if {Simple} rather than {Coord}, is realised in the PROCESS function. That is every simple clause has a main verb as an immediate constituent. In general, then, Ftr rules derive an unordered function set from the item's feature-set, thus:



The second stage of the derivation is to re-arrange this function set into an ordered set of function-bundles, by means of structure-building, or SB, rules. These rules are explained more fully in section 4; for the moment we diagram their effect as shown on the next page.



Each function-bundle corresponds to an immediate constituent of the clause and determines at least some of the constituent's features. This relation of function to feature is also a realisation relation, and the rules of the grammar which govern the derivation are "function realisation", or FNr, rules. We may add the application of the clause FNr rules to the diagram, as shown on the next page:



This diagram oversimplifies the feature and function sets involved in making a clause. In particular it does not make clear that the realisation relation is only rarely one-to-one. The rules and the worked examples given in sections 5 to 9 of this chapter make clear what the relation is in every case, and a comprehensive worked example is given in Appendix 2.

5.3 Realisation relations

All realisation rules are written according to the same formation constraints, whether they are Ftr or FNr rules. In what follows, feature names will continue to appear in

miniscule with a capital initial, and function names in capitals. This helps us to see what the rules are doing, but is a distinction not made in the computer program text.

All realisation rules mention a realisate and its realisation. In the simplest case there is just one realisation, and it is unconditional. So the clause Ftr rule

{Bound {+BINDER}}

means that every clause with the feature {Bound} contains an immediate constituent which has the function BINDER. Likewise the clause FNr rule

{BINDER {+Word}}

means that the constituent which has the BINDER function is an item with the feature {Word}. Very often the realisate is not a single element; in fact the last example must really be

{BINDER {+Word} {+Binder}}

because we know that the BINDER function must be done by a word classified as a {Binder} in system 113.

Most realisation rules are a little more complicated than this because the realisation is not unconditional. We may therefore add to each realisation a condition, either positive or negative. The condition always refers to another element in the set from which the realisate is drawn, so that a condition in an Ftr rule always refers to a feature and in an FNr rule to a function. For example the first clause Ftr rule is:

{Clause {+PROCESS IFF {Simple}}}

This expresses the rule that we have mentioned before, namely that every clause item, provided it is simple not coordinate, contains an immediate constituent with the function PROCESS. We may mention any number of elements in the condition, as

{Realisate {+REALISATION IFF {A B C}}}

The condition in this example is satisfied if any of A or B or C is present in the set from which Realisate is drawn. So far we have examined positive conditions. Negative conditions are written as in

{PREPREL {+Prepg IFF {-DANGLINGP}}}

which means that the item which has the function PREPREL has the feature {Prepg} only if it does not also have the function DANGLINGP. The negative sign reverses the truth-value of the condition as a whole, so

{-A B C}

is true only if neither A nor B nor C is present. Thus we have

{Eff {+GOAL IFF {-Gintrans Ustandg}}}

This clause Ftr rule tells us that an {Effective} clause, system 10, contains an immediate constituent with the function GOAL, but only if the clause is neither goal-intransitive, system 12, nor understanding its goal, system 37. Finally, we are not confined to just one condition per realisation, but can add conditions linked by AND or OR:

{Realisate {+Realisation IFF {A} OR {-B}}}

We said in the last section that the Ftr rules produce an unordered function set: that simple statement is in one respect untrue, since the Ftr rules may have a realisation which is not a solo function but is a function-bundle. So we have

{Receptive {+SUBJECT = GOAL IFF {Eff}}}

This means that a {Receptive} clause, provided it is {Effective}, contains an immediate constituent which has the SUBJECT and GOAL functions. We notice that there is no '+' before GOAL. The reason is that the set of Ftr rules, like all the other rule sets, is ordered, and an earlier Ftr rule has already required the GOAL function as realisation. The present rule therefore requires that the SUBJECT function be present, and that it be bundled with the already existing GOAL. Ftr rules permit any or none of the functions thus bundled to be ones already required, and so some Ftr rules simple put functions into bundles:

{Goalrel {REL = GOAL IFF...}}

When this rule is reached, both REL and GOAL have already been required as the realisation of other features; this rule says that, provided the condition is satisfied, the two go in a bundle.

5.4 Structure-building rules

There are two sorts of SB rule; their tasks overlap to some extent, and they could be amalgamated into a single type, but it is convenient to distinguish them. Type 1, which we shall call SB1 rules, add functions to the set

resulting from the application of the Ftr rules. These rules are invariably conditional, and the condition is the presence in, or the absence from, the unordered function set of a function or functions. The condition in an Ftr rule, on the other hand, mentions only features. Although this type of SB1 rule could be replaced by complex Ftr rules, it is retained for its advantage of clarity and simplicity. SB1 rules may also put functions together in a bundle, whether these functions have just been added or were introduced by Ftr rules. An example of a clause SB1 rule is:

{+EN = PROCESS IFF {PASSIVE PERFECT}}

The conventions are much as for realisation rules. This example means that if any clause constituent has the PASSIVE or PERFECT function, the EN function must be introduced and put in a bundle with the PROCESS. The EN function, of course, constrains the item which has it to be a past participle; its FNr rule is:

{EN {+Participle} {+Pastpart}}

The set of function-bundles and functions, which we may regard as unary bundles, remains unordered after the application of the SB1 rules. The SB2 rules now order the set and perform further amalgamations of function-bundles. Their format is different from that of all other rules. The inequality sign is used to specify order, so:

{ENVIR > COMMA}

means that the ENVIR function, and any function bundled

with it, goes before the COMMA function and its bundle. The interpretation given to this rule should be carefully noted, because the ability to order function-bundles by ordering representative functions from each is of great practical importance in making a grammar which works. Any number of functions can be mentioned in this way: the rule which settles the order of auxiliary verbs in a clause begins:

{MODAL > PERFECT > PASSIVE...}

It probably goes without saying that these rules are opportunistic, applying first to whichever of the functions mentioned are present, so that if the auxiliaries include modal and passive, but not perfect, as in:

"You should be rewarded"

the rule above still arranges the order correctly.

The SB2 rules include two further facilities. If we write {ENVIR > {PREPREL REL...} > SUBJECT...} we mean that ENVIR precedes whichever of PREPREL, REL,... occurs, and that precedes SUBJECT. The contents of the internal bracket, PREPREL, REL..., are exclusive alternatives, conveniently mentioned en bloc. Secondly, wherever we may write ">" we may write " = " with the natural interpretation. So

{FINITE = PAST = {MODAL PERFECT...}}

means that FINITE and its bundle are amalgamated with PAST and its bundle, and likewise with the first function listed in the internal bracket which is present. We shall very shortly come to some examples.

5.5 Clause Rules

This section first sets out the Ftr, the SB, and the FNr rules for clause items; these rules are mainly for reference and should not be examined in detail at first reading. The rules are followed by some explanatory notes, and worked examples of the cycle of derivation.

5.5.1 Clause Ftr rules

The Ftr rules for Clause features are:

```
      {
1     {Clause          {+PROCESS IFF {Simple}}}
2     {Eff             {+GOAL IFF {-Gintrans Ustandg}}
                       {+ACTOR IFF {-Ustanda}}}
3     {Descr           {+ACTOR IFF {-Ustands}}
4     {Op              {+INITR = +SUBJECT IFF {-Ustandg}}
                       {ACTOR = SUBJECT IFF {Eff} AND
                        {-Ustanda}}}
5     {Mid             {INITR = +SUBJECT = ACTOR IFF
                        {-Ustandi}}
                       {INITR = GOAL IFF {-Ustandi Ustandg
                        Descr}}}
6     {Receptiv        {+SUBJECT = GOAL iff {Eff}}
                       {+SUBJECT = ACTOR IFF {Descr}}
```

```
                    {+PASSIVE}}
7       {Indic      {+FINITE}}
8       {Interrog   {+MFOC}}
9       {Modal      {+MODAL}}
10      {Modalfut   {+MODALFUT = MODAL}}
11      {Modalpos   {+MODALPOS = MODAL}}
12      {Past       {+PAST}}
13      {Finite     {+FINITE}}
14      {Bound      {+BINDER}}
15      {Ifbound    {+IFB = BINDER}}
16      {Thobound   {+THO = BINDER}}
17      {Preprel    {+PREPREL IFF {Explic}}}
18      {Nomrel     {+REL IFF {Explic}}}
19      {Dangling   {+DANGLINGP}}
20      {Goalrel    {REL = GOAL IFF {Explic} AND {Ustandg}}}
21      {Infin      {+INFIN = PROCESS}}
22      {Participl  {+PARTICIPLE = PROCESS}}
23      {Ing        {+ING = PARTICIPLE}}
24      {Perfectiv  {+PERFECT}}
25      {Imminent   {+IMMINENT}
                    {+INFIN = PROCESS}}
26      {Enviradj   {+ENVIR = CLAUSE}}
27      {Ifadj      {+HYPOTH = ENVIR}}
28      {Thoadj     {+CONCESS = ENVIR}}
29      {Appadj     {+APPENDIX}}
30      {Accompadj  {+WITHOBJ = APPENDIX}}
31      {Methodadj  {+BYOBJ = APPENDIX}}
32      {Timeadj    {+TIME = APPENDIX}}
```

```
33    {Tfirst          {+FIRST = TIME}}
34    {Tyet            {+YET = TIME}}
35    {Pradj           {+RESTRICT = +ADVERB}}
36    {Prjust          {+JUST = ADVERB}}
37    {Pryet           {+YET = ADVERB}}
38    {Negative        {+NEG = ADVERB}}
      }
```

SB1 rules

```
      {
1     {+COMMA IFF {ENVIR}}
2     {+EN = PROCESS IFF {PASSIVE PERFECT}}.
3     {+POSTVERB = ACTOR IFF {ACTOR} AND {-ACTOR=SUBJECT}
        AND {-ACTOR=REL}}
4     {+POSTVERB = GOAL IFF {GOAL} and {-GOAL=SUBJECT}
        AND {-GOAL=REL}}
5     {POSTVERB = DANGLINGP IFF {POSTVERB} AND {DANGLINGP}}
      }
```

SB2 rules

```
      {
1     {ENVIR > {PREPREL REL MFOC BINDER} > SUBJECT >
       PROCESS > POSTVERB > APPENDIX}
2     {FINITE = PAST = MFOC = {MODAL PERFECT PASSIVE
       IMMINENT PROCESS}}
3     {MODAL > PERFECT > PASSIVE > IMMINENT > PROCESS}
4     {FINITE > RESTRICT}
5     {ENVIR > COMMA}
      }
```

5.5.3 Clause FNr rules

```
1   {ADVERB          {+ Adverb} {+ Word}}
2   {APPENDIX        {+ Prepg  IFF {BYOBJ} OR {WITHOBJ}}
                     {+ Adverb  IFF {TIME}}
                     {+ Word  IFF {TIME}}
3   {BINDER          {+ Word}  {+ Binder}}
4   {BYOBJ           {+ By}}
5   {CLAUSE          {+ Clause}}
6   {COMMA           {+ Word}  {+ Link}  {+ Comma}}
7   {CONCESS         {+ Thobound}}
8   {DANGLINGP       {+ Danglingp}}
9   {EN              {+ Participle} {+ Pastpart}}
10  {ENVIR           {+ Dependent}  {+ Finite} {+ Bound}}
11  {FINITE          {+ Tensed}}
                     {+ Remote IFF  {PAST}    {+ Present
                        IFF {-PAST}}}
12  {FIRST           {+ First}}
13  {HYPOTH          {+ Ifbound}}
14  {IFB             {+ If}}
15  {IMMINENT        {+ Vg}  {+ Grammatical} {+ Begoing}}
16  {INFIN           {+ Infinitive}  {+ Tϕ}}
17  {ING             {+ Participle}  {+ Prespart}}
18  {NEG             {+ Negative}}
19  {N1              {+ 1}}
20  {N2              {+ 2}}
21  {N3              {+ 3}}
22  {MODAL           {+ Vg} {+ Grammatical}  {+ Modal}}
```

```
23 | {MODALFUT          {+ Will}}
24 | {MODALPOS          {+ Can}}
25 | {PASSIVE           {+ Vg}  {+ Grammatical} {+ Aspectual}
                         {+Be}}
26 | {PERFECT           {+ Vg}  {+ Grammatical} {+ Aspectual}
                         {+ Have}}
27 | {POSTVERB          {+ Object} {+ Ng}}
28 | {PREPREL           {+ Prepg IFF   {DANGLINGP}}
                        {+ OF  IFF  {DANGLINGP}}
                        {+ Ng  IFF  {DANGLINGP}}  {+ Relative}}
29 | {JUST              {+ Just}}
30 | {PROCESS           {+ Vg}  {+ Lexical}}
31 | {REL               {+ Ng}  {+ Relative}}
32 | {SUBJECT           {+ Ng}  {+ Subject}
33 | {THO               {+ Although}}
34 | {WITHOBJ           {+ With}}
35 | {YET               {+ Yet}}

   | }
```

5.5.4 Notes on the rules

This body of rules forms an organic whole and is best understood by being put to work. This subsection therefore offers only brief explanatory comments, and the next comprises several worked examples.

The Ftr rules

Rules 2 - 6 are concerned with the {TRANSITIVITY}sub-

network and related functions. Taken with the description of those systems in the last chapter, they are self-explanatory. We note only that the features {Ustanda} {Ustandg}, and {Ustandi} are abbreviations for the features marked in the {UNDERSTANDING} sub-network.

Rules 7 -23 relate to the {MOOD} systems. In rules 15 and 16 a function is used to make sure that the right subordinating conjunction is used. In these and other such cases, what is perhaps a lexical decision is taken within the grammar. However, such selections within the closed-class dictionary are regarded here as essentially grammatical, as much as are the selections of number, person, and gender of a pronoun.

Rule 19 introduces DANGLINGP as a function. This function gives the item which has it the dominance of the preceding relative and hence licences a dangling preposition: the relative may, of course, be implicit:

"the pie she burnt the top of".

If the preposition is fronted, not dangling, as:

"the pie of which she burnt the top"

the item "the top" does not have the function DANGLINGP since there is nothing in the item's structure to indicate its relation to the relative.
So we distinguish

"the shop in which you like the kippers"

where "the kippers in the shop" is the referent, the object of your liking, from:

"the shop in which you got the kippers"

where "the kippers" which were got were presumably not all "the kippers in the shop", on semantic grounds not formalised in the grammar.

In rules 17 and 18 two different relative functions are introduced. PREPREL and REL are distinguished because whereas REL will be bundled with ACTOR, INITIATOR, or GOAL, the three PROCESS participants, PREPREL cannot be. Hence they must be distinguished even though the constituent which has either function may be the same in each case. Thus

"The match which you lighted"

might be completed by "---your pipe with". In the shorter version"which" has the functions REL and GOAL, in the longer PREPREL.

Rules 24 and 25 are related to the {ASPECT} systems, and 26-38 to the {ADJUNCT} systems. We noticed in the last chapter the unsatisfactory character of the {ADJUNCT} systems, and these present rules are correspondingly in need of reformulation.

The SB rules

The character and motivation of these rules will

become clearer in the next subsection. Here we note only the functions MFOC and POSTVERB. These two functions are really place-markers, or buoys, in the developing string of function bundles: as we shall see, other functions are moored to these buoys, until finally a complete bundle is constructed around them. MFOC, representing "mood-focus", is introduced by Ftr rule 8, when the clause is {Interrogative} . Its task is to mark the site at which the constituent signalling the interrogative mood will appear. The present program produces only polar interrogatives, and we could rewrite the rules to manage without MFOC, but a grammar which accounted for interrogative "wh -" words would be simplified by it, and we keep it here for consistency. The function POSTVERB is likewise a position marker, the site for the first process participant following the verb. A more complete grammar requires many more such buoy functions, and we consider at the end of this chapter whether they constitute a drawback of this type of grammar.

The FNr rules

These rules are simple and are best explained by the worked examples which follow.

5.5.5 Examples

Even within the present little grammar, clauses are too various to be exhaustively exemplified. We give two

examples here, and the rules are sufficiently simple that further examples can easily be tested as required.

Example 1.

We suppose that the program is about to construct the sentence

"You blocked me".

This comprises a single clause item, and we shall examine the cycle of rule applications which derive from its feature list a feature description of each immediate constituent of it.

The clause item's feature set is:

{Clause Simple Eff Ext Op Gtrans Past Declar Indic Indep}

To this set we first apply the clause Ftr rules as follows

Rule No:	Result of Application:
1	+ PROCESS
2	+ GOAL + ACTOR
4 a	+ INITR = + SUBJECT
b	(ACTOR = SUBJECT = INITR)
7	+ FINITE
12	+ PAST

This yields the unordered set:

(PROCESS GOAL (ACTOR = SUBJECT = INITR) FINITE PAST)

Applying the SB1 rules to this set, we obtain

4 (GOAL = POSTVERB)

At this stage the unordered set is:

((GOAL = POSTVERB) PROCESS (ACTOR = SUBJECT = INITR) FINITE PAST)

and we apply the SB2 rules to conflate and order the set members:

1 (SUBJECT =ACTOR=INITR) > PROCESS > (POSTVERB=GOAL)

Notice how we were able to order bundles of functions by mentioning just one representative of each bundle in the rule.

2 (FINITE = PAST = PROCESS)

The ordered set of function bundles is thus:

((SUBJECT = ACTOR = INITR) > (FINITE = PAST = PROCESS) > (POSTVERB = GOAL))

Each bundle corresponds with an immediate constituent of the clause, and by applying the FNr rules to each bundle, we obtain the features which realise the functions. Thus we have for (SUBJECT = ACTOR = INITR)

32 { + Ng} {+ Subject}

and we all know about the first constituent is that its features include {Ng Subject}. For the second bundle we have (FINITE = PAST = PROCESS), and so

11 {+ Tensed} {+ Remote}

30 {+ Vg} {+ Lexical}

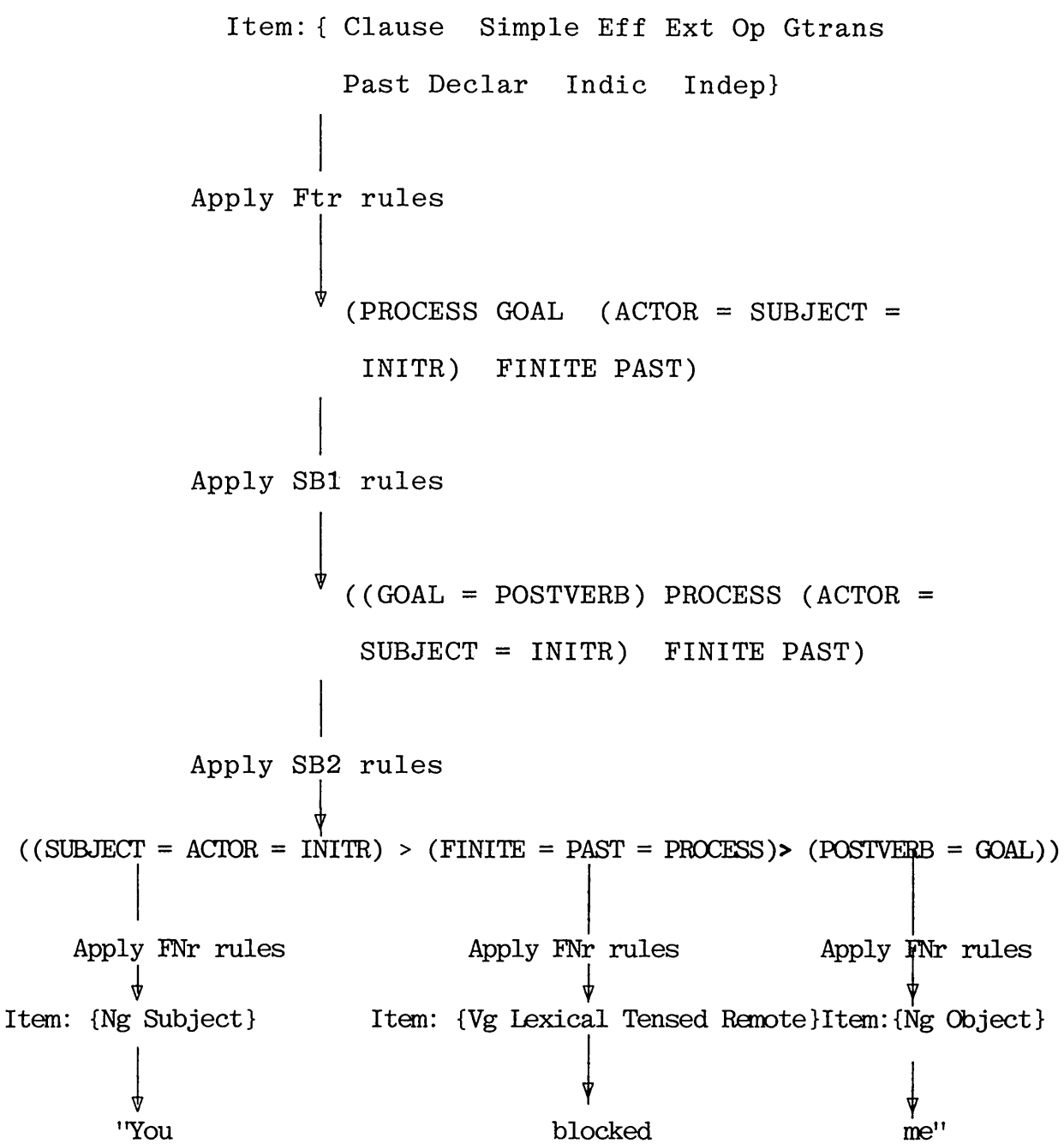
So we know that the second constituent is{Vg Lexical Tensed Remote}. Likewise for the third:

(POSTVERB = GOAL)

27 {+ Object} {+ Ng}

yielding {Ng Object}.

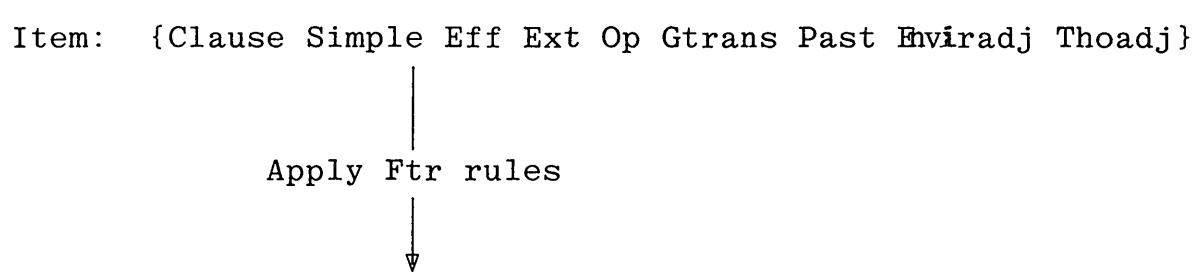
We can sum up this example in the following diagram:-

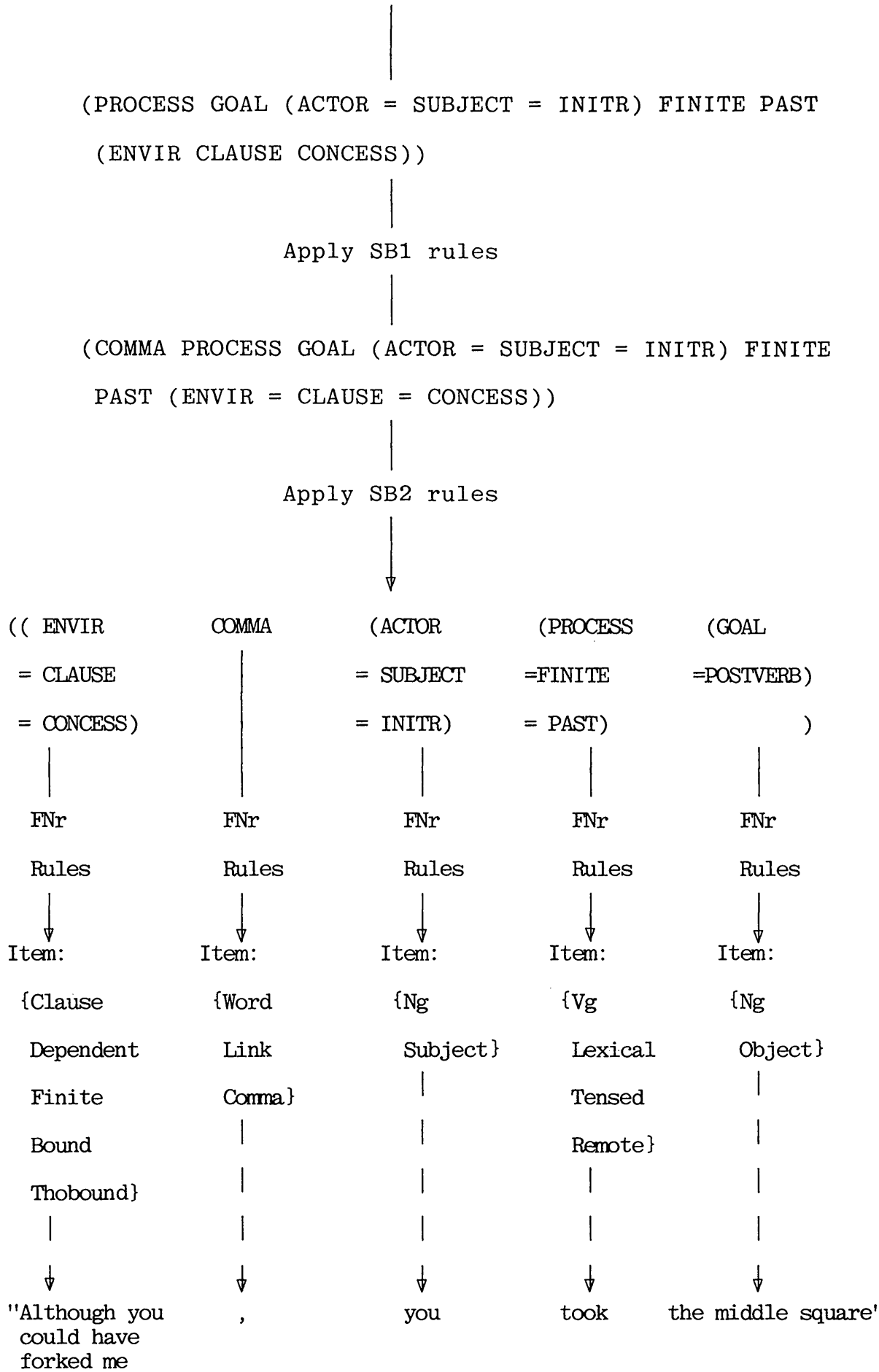


This diagram is intended solely to illustrate the operation of the clause Ftr, SB and FNr rules of the grammar. It says nothing about the source of the clause item's features, nor about the operations responsible for the appearance of the quoted words at the bottom of the constituent structure tree. However, as we saw in section 1.2 of chapter 4, the general principle is that each specialist constructor procedure works from a partial feature list and a semantic representation of the item to a partial or complete feature list for each constituent. The source of the representation and features of the clause is explained in section 3 of chapter 6, and is illustrated in Appendix B. The grammatical rules used in constructing Noun-groups and Verb-groups are explained in the next two sections of this chapter, and their constructor specialists are designed in section 7 and 6 respectively of chapter 6.

As our second example we take a more complex item, but we present it only in the summary form adoped for the recapitulation of the last example. The item we are going to construct is:

"Although you could have forked me, you took the middle square".




(PROCESS GOAL (ACTOR = SUBJECT = INITR) FINITE PAST
(ENVIR CLAUSE CONCESS))
Apply SB1 rules
(COMMA PROCESS GOAL (ACTOR = SUBJECT = INITR) FINITE
PAST (ENVIR = CLAUSE = CONCESS))
Apply SB2 rules
((ENVIR = CLAUSE = CONCESS)
COMMA
(ACTOR = SUBJECT = INITR)
(PROCESS =FINITE = PAST)
(GOAL =POSTVERB))
FNr Rules
Item:
{Clause Dependent Finite Bound Thobound}
{Word Link Comma}
{Ng Subject}
{Vg Lexical Tensed Remote}
{Ng Object}
"Although you could have forked me
,
you
took
the middle square"

5.6 Verb Groups

As we said in section 5.1 of the last chapter, the Vg systems classify the Vg item in just enough detail for procedures not specified in the grammar to get the right verb in each case. These procedures are described in chapter 6, section 6. So there is no realisation of functions within the Vg, since the verb is realised as an immediate constituent of the clause.

5.7.1 Noun Group Ftr rules

1	{Ng	{+ HEAD}}
2	{Clause	{+ CLAUSE}}
3	{Sg	{+ SINGULAR = HEAD}}
4	{Pl	{+ PLURAL = HEAD}}
5	{Def	{+ DEF}}
6	{Indef	{+ INDEF}}
7	{Numbered	{+ CARD}}
8	{Modified	{+ MODIFIER}}
9	{Adjgqual	{+ QUALIFIER = + ADJG}}
10	{Clausequal	{+ QUALIFIER = + RCLAUSE}}
11	{Prepgqual	{+ QUALIFIER = + PREPG}}
12	{Danglingp	{+ DANGLINGP = PREPG}}
13	{Noun	{+ CLASS}}
14	{Propernoun	{+ NAME}}
15	{Pronoun	{+ PRON}}
16	{Prostring	{+ ANAPHORS = PRON}}
17	{Proref	{+ ANAPHORR = PRON}}

```
18 | {1                  {+ ADRESSER}}
19 | {2                  {+ ADRESSEE}}
20 | {3                  {+ REFEREE}}
21 | {An                 {+ LIFE = HEAD}}
22 | {Fem                {+ FEM = HEAD}}
23 | {Determined         {+ DET}}
24 | {Deictic            {+ DEICTIC = ANAPHORR}}
25 | {Relative           {+ REL = PRON}}
26 | {Remote             {+ REMOTE = DEICTIC}}
27 | {Nomposs            {+ NG = DET}}
28 | {Adjposs            {+ ADJG = DET}}
29 | {Object             {+ ACCUSATIVE = HEAD}}
30 | {Genitive           {+ GENMKR}}
31a {Possess            {+ DET IFF {- DET PROREF}}
  b                     {+ GENITIVE = DET IFF {-PROREF}}
  c                     {+ GENITIVE = ANAPHORR IFF {PROREF}}}
    }
```

5.7.2 Noun-Group SB rules

SB1 rules

```
  | {
1 | {+ DET IFF {DEF} AND {- DET ANAPHORR NAME}}
2 | {+ DET IFF {INDEF} AND {ANAPHORS} AND {MODIFIER}}
3 | {+ DET IFF {INDEF  AND {-ANAPHORS DET CARD PLURAL}}
4 | {HEAD  = {CLAUSE PRON NAME CLASS MODIFIER CARD}}
5 | {HEAD  = {ADRESSER  ADRESSEE REFEREE}}
6 | {+ OFOBJ  =  PREPG IFF { PREPG  AND  {PREPG = QUALIFIER}}
  |  }
```

SB2 rules

{

1 {INDEF = {DET HEAD}}

2 {DEF = {DET ANAPHORR NAME}}

3 {DET > CARD > MODIFIER > {CLAUSE CLASS NAME PRON} >
ADJG > PREPG > RCLAUSE > GENMKR}

}

5.7.3 Noun-group FNr rules

```
1    {ACCUSATIVE      {+Accusative}}
2    {ADRESSER        {+1}}
3    {ADRESSEE        {+2}}
4a   {ADJG            {+Adjg}
 b                    {+Complex IFF {QUALIFIER}}}
5a   {ANAPHORR        {+Definite IFF {DEF}}
 b                    {+Indef IFF {-DEF}}
 c                    {+Inan IFF {-LIFE}}}
6    {ANAPHORS        {+Indef}}
7    {CARD            {+Word} {+Number}}
8    {CLASS           {+Word} {+Noun} {+Common}}
9    {CLAUSE          {+Clause} {+Dependent}}
10   {DEF             {+Definite IFF {DET}}
11   {DEICTIC         {+Demonstrative}}
12a  {DET             {+Word IFF {-HEAD DEICTIC NG ADJG
                       GENITIVE}}
  b                  {+Article IFF {-HEAD DEICTIC NG ADJG}}}
13a  {GENITIVE        {+Genitive IFF {NG}}
  b                  {+Possess IFF {-NG}}}
14   {GENMKR          {+Word} {+S}}
```

```
15   {INDEF          {+Indef IFF {DET}}
16   {LIFE           {+An}}
17   {MODIFIER       {+Adjg} {+Simple}}
18   {NAME           {+Word} {+Noun} {+Proper}}
19   {NG             {+Ng}}
20   {OFOBJ          {+Of}}
21   {PLURAL         {+Pl}}
22   {PREPG          {+Prepg} {+Danglingp IFF {DANGLINGP}}}
23   {PRON           {+Word} {+Pronoun}}
24   {RCLAUSE        {+Clause} {+Dependent} {+Finite}
                     {+Relative}}
25   {REFEREE        {+3}}
26   {REL            {+Relative}}
27   {REMOTE         {+Remote}}
28   {SINGULAR       {+Sg}}
     }
```

5.7.4 Noun-group rules - notes.

The Ftr rules

Rule 1 ensures that every Ng has a HEAD. This function, like MFOC and POSTVERB in the clause rules, is a buoy. Ftr rules 3 and 4, together with SB1 rules 4 and 5 and SB2 rule 1, show other functions being moored to HEAD, and the examples in the next section demonstrate its use.

Rules 5 and 6 relate to the features in system 64, which were explained in section 5.2 of the last chapter. There we saw that the definiteness or otherwise of a noun-group was analysed in abstraction from the particular constituents involved. These Ftr rules therefore introduce functions which the SB rule will put into the right bundles.

Rules 18,19, and 20, together with SB1 rule 5, ensure that the person of the noun-group is correctly reflected

in the head word of the noun-group. The rules are redundant, of course except when the word is a definite pronoun ("I, you, he, ---") since only the definite pronoun needs a different form for different persons. They are used in every Ng however for programming convenience. To snuff out a possible source of confusion, it should perhaps be mentioned that these functions have no connection with the selection of the correct person in possessive pronouns and possessive adjectives. Obviously "mine" in:

"(You blocked) mine"

is a 3rd person Ng, and the head word of it, "mine" has the REFEREE function, but equally "mine" is a first-person possessive as against "yours, his,---". Likewise in "my line", the "line" constituent has the REFEREE function, while "my" is a first-person possessive adjective. The program's constructor functions treat the selection of "mine" against "yours, his ---", and of "my---" against "your ---, his---, ---" as a lexical matter comparable with the selection of "adjacent" against "opposite". They do, however, specify the word they want by mentioning a feature-list to the closed-class dictionary. This means that the source of the person-feature in the feature list specifying a definite pronoun such as "you" is not the same as the source of the corresponding feature in the specification of a possessive adjective or pronoun. In the former case the feature is derived via the rules we are examining, while

in the latter its source is a semantic specialist within a construction procedure.

Rule 24 is somewhat ad hoc. It works only because in the program's output the deictic function is always performed by a pronoun. A better analysis might place systems 67, 72, and 68, which analyse for deixis, in parallel with system 64, which analyses for {Definite} or {Indefinite}. In other words, it might be better to analyse deixis in abstraction from the particular constituent which conveys it, and to rely upon the SB rules to allot the function to its correct function bundle, just as we do with DEF and INDEF.

The SB rules

The first three SB1 rules introduce a determiner function if necessary: the function thus introduced will be performed by an article, "a" or "the". Rule 4 marks the head constituent of the noun-group simply by attaching HEAD, and its associated bundle, to that function whose bundle will be placed by the SB2 rules in the right-most position in the string before any QUALIFIER or GENMKR. Rule 5 completes the HEAD bundle: this rule is a good illustration of the use of a buoy function, since in the absence of HEAD the rule would have to be written to take account of all the possible functions which could be in the head bundle.

Rule 6 is another ad hoc short cut. It happens that the program's output includes only those preposition-group qualifiers which are introduced by "of" as:

"one of your edges" or "the middle of the board"

So this rule marks every QUALIFIER which is a PREPG ad an OFOBJ. Obviously a revision of the {ADJUNCT} systems would substitute an adequate analysis for this simple assumption.

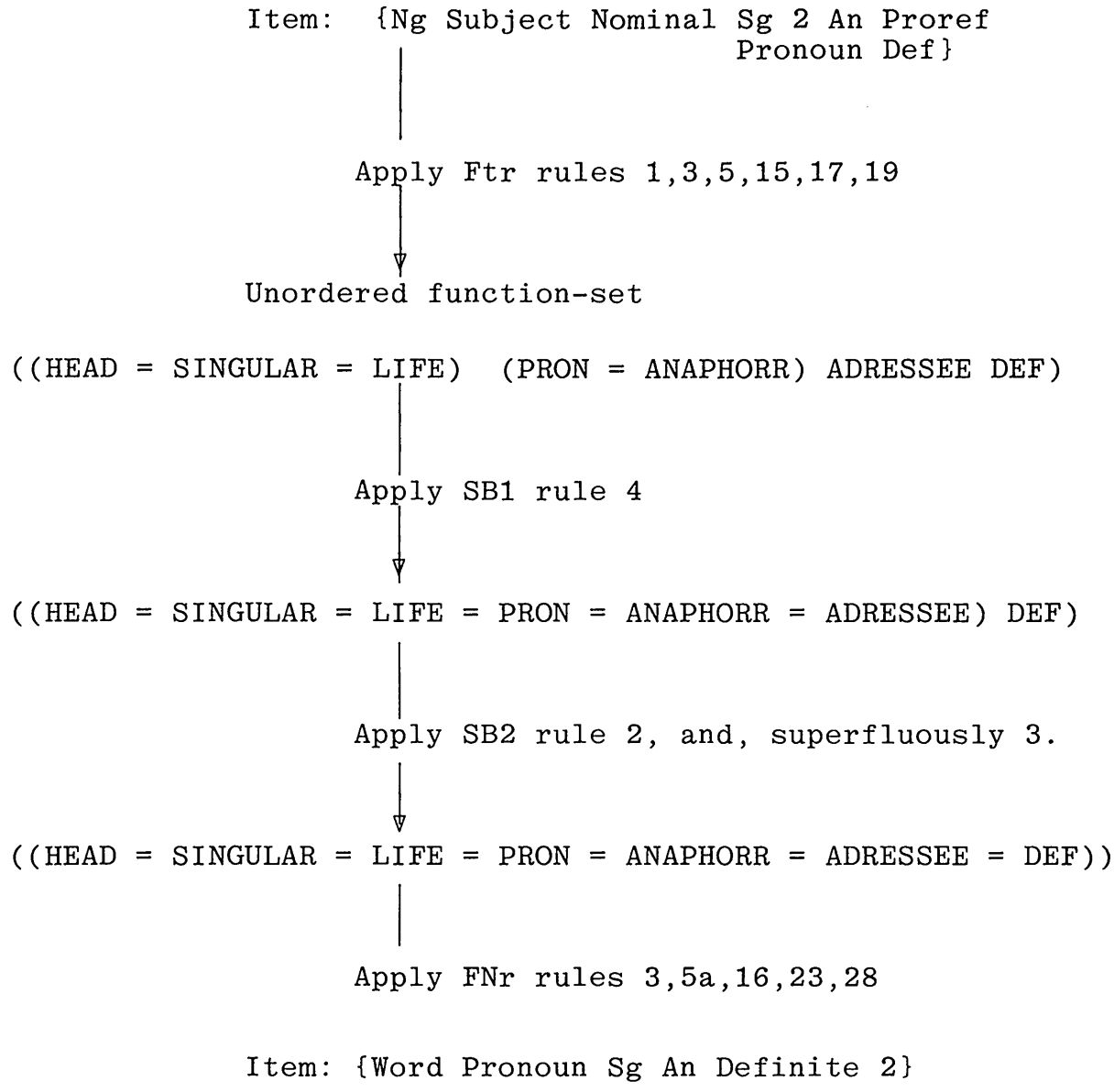
The SB2 rules are simple and best explained by demonstration.

5.7.5. Noun Group examples

The examples in this group are presented in dummary form, but at each application of the rules appears a note of the rules applied, from which the details may be reconstructed without difficulty.

Example 1

In this example, the system is constructing the noun-group "you", as subject.

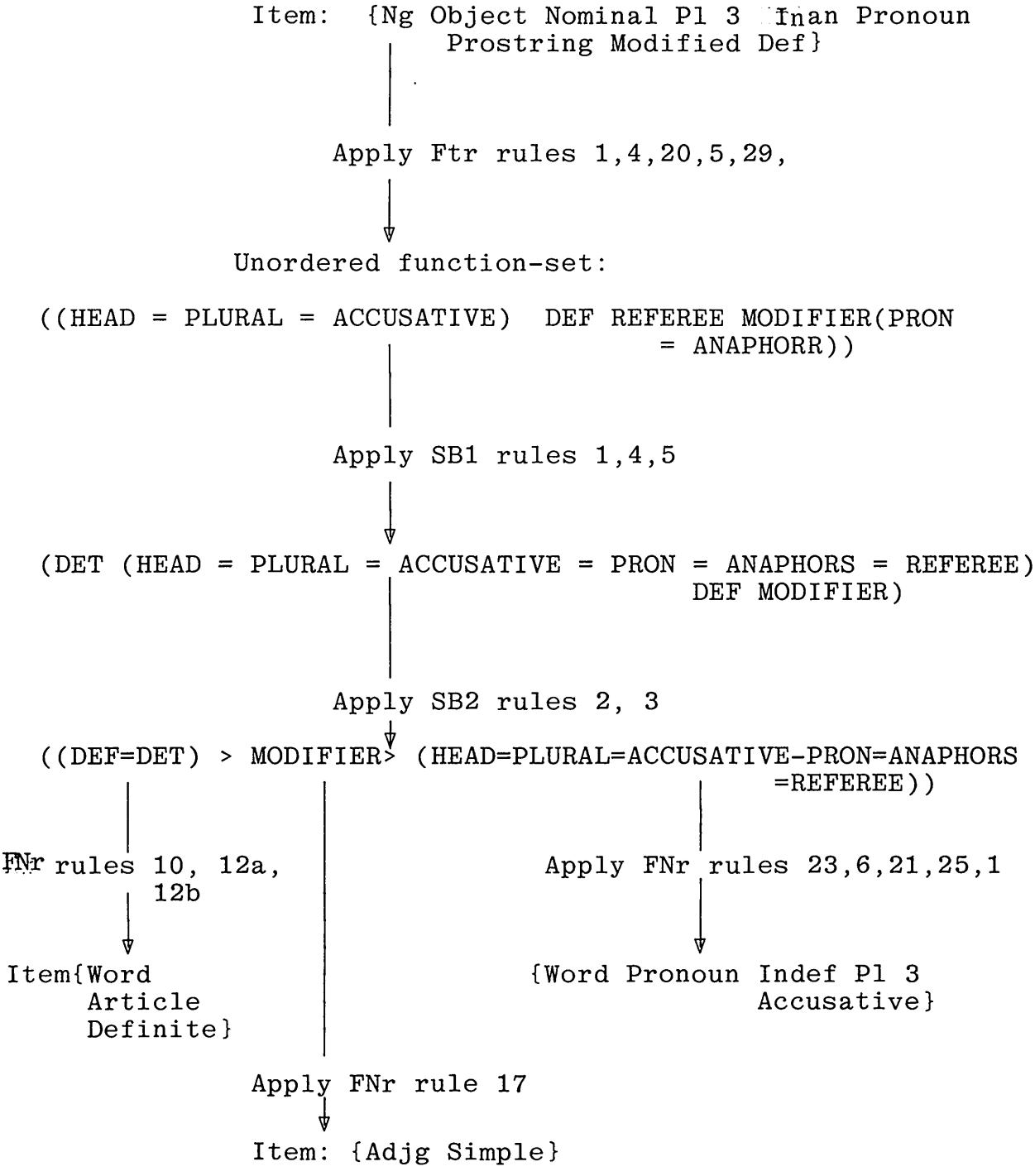


The final feature list, it will be observed, completely specifies the pronoun "you" in the Word systems subnetwork. This is a very simple example, but it demonstrates the way the HEAD function is used as a collecting-point for other functions, and the way DEF is independently motivated and finally attached to the right bundle.

Example 2

In this example the system is constructing "the other ones", as in

"(--- and you took one of) the other ones"



5.8 Preposition-group rules

The preposition-group is a simple structure, and the rules given here are correspondingly simple.

5.8.1 Ftr rules

```
     {
1    {Prepg              {+NOM IFF {-Danglingp}}
2    {Relative           {+RELATIVE = NOM}}
3    {Between            {+PREP = +BETWEEN}}
4    {By                 {+PREP = +BY}}
5    {Of                 {+PREP = +OF}}
6    {To                 {+PREP = +TO}}
7    {With               {+PREP = +WITH}}
     }
```

SB2 rules

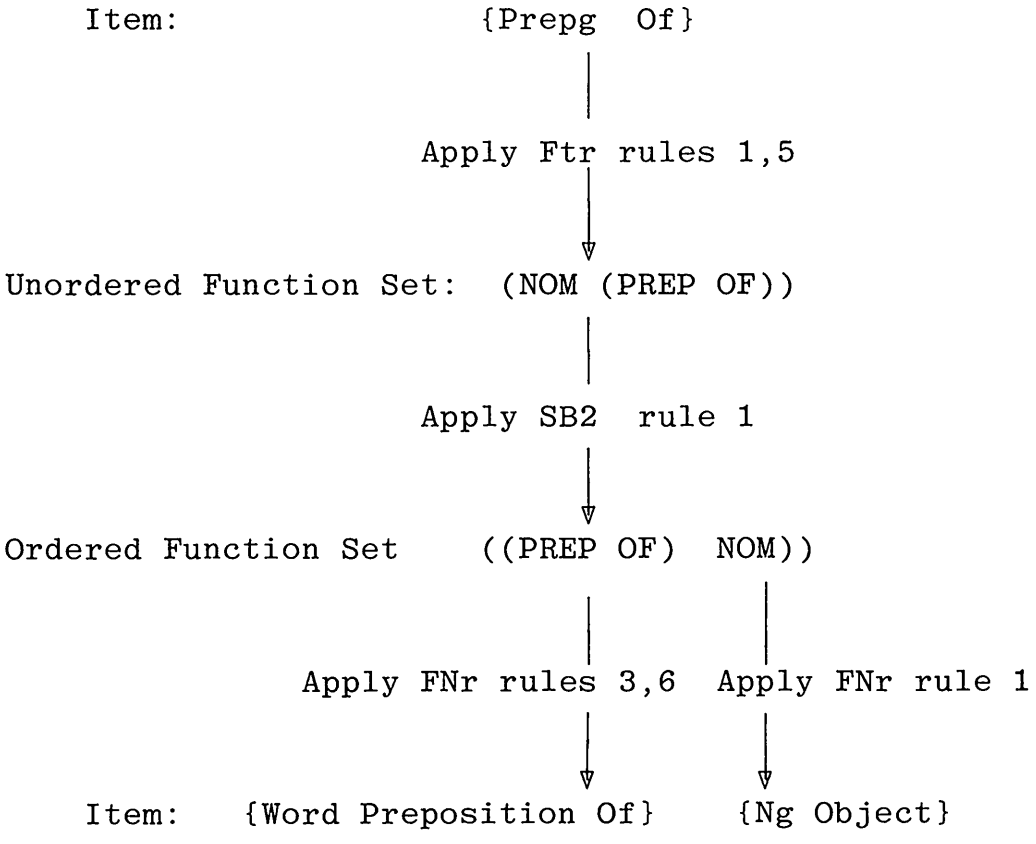
```
1    {{PREP > NOM}}
```

FNr rules

```
     {
1    {NOM                {+Ng} {+Object}}
2    {RELATIVE           {+Relative}}
3    {PREP               {+Word} {+Preposition}}
4    {BETWEEN            {+Between}}
5    {BY                 {+By}}
6    {OF                 {+Of}}
7    {TO                 {+To}}
8    {WITH               {+With}}
     }
```

5.8.2 An Example

These rules are very simple and need no commentary: section 5.3 of the last chapter explained why the preposition-word was made a feature of the Prepg. The following example illustrates their use: the item being constructed is "(---one) of the other ones".



5.9 Adjective-Group rules

The following rules suffice for the program's output:

Adjg Ftr rules

{

1 {Adjg {+MOD}}

2 {Complex {+FOCUS}}

3 {Ngfocus {+NOM = FOCUS}}

4 {Pgfocus {+PREP = FOCUS}}

}

SB2 rules

1 {MOD > FOCUS}

FNr rules

1 {MOD {+Word} {+Adjective} {+Lexical}}

2 {NOM {+Ng} {+Object}}

3 {PREP {+Prepg}}

}

These rules are simple enough to be clear without a commentary or examples. However, we may note that MOD is realised in {...Lexical} adjectives only, because {Deictic} and {Possessive} adjectives are regarded as determiners: they do not have any of the options analysed in the Adjg systems set out in the last chapter, and are therefore treated as immediate constituents of the Ng, not of an Adjg.

5.10 Doing without functions

The feature-set of an item is realised in a function-set, and the members of each function-set are in turn realised in features of the corresponding immediate constituent of the item. So perhaps we could manage without functions, as Hudson (1972b:9) suggests. Perhaps we could find rules which would map immediately from the item's feature-set to an ordered set of feature-sets corresponding to the item's immediate constituents.

An example of the proposed revision concerns the SUBJECT function. The subject of a clause must be a nominal or clausal noun-group, with the features {Ng Nominal...} or {Ng Clause...}. But the object must likewise be a nominal or clausal noun-group, and so in the present grammar we distinguish the nominative subject by {...Subject...} from the accusative, or {...Object...} object. In the proposed revision, on the other hand, such features can hardly be employed since that would be simply to smuggle back the banished functions disguised as features. Instead it is proposed that we mark features of the immediate constituent to show what features of the parent item they realise: these dependencies will uniquely identify the constituent. In the present case we know that every indicative clause must have a subject, so the subject noun-group is

dependent upon the {Clause ---} feature of the parent item. We shall write this dependency {Ng {Clause} ---} which means that the feature {Ng} realises the feature {Clause} in the parent item's feature set. Thus if the constituent has the feature-set

{Ng {Clause} Nominal Sg Def ---&c}

we know that it is the subject.

We know, of course, that a clause may have other immediate constituent noun-groups, and in particular an object. But the object is required not because the parent item is a clause, but because the parent is a clause with certain transitivity characteristics which we may for the moment write as the feature {Two-Participant} . Therefore the object's dependency is {Ng {Two-Participant} ---} and if a constituent had the feature-set

{Ng {Two-Participant} Nominal Sg Def ---&c}

we would know that it was the object. If we can thus manage without functions in the theory, we should surely do so.

This argument is reinforced by experience in constructing a systemic functional grammar, even one as simple as the present one. The grammarian finds himself forced to invent a large number of functions in order to capture every variation of immediate

constituent structure and every environmental constraint upon a constituent. "Function" in the grammar has a precisely defined meaning, and each of the functions he has to postulate is indeed a function in the sense defined. However, the term "function" was chosen in the conviction that every variation and constraint captured by the function notation was indeed reflected in a difference in the 'job being done' by the constituent in question. We are therefore likely to be disillusioned if we find that to make the grammar adequately generative we have to postulate many functions which seem unlikely candidates for 'jobs being done'.

This is to some extent a problem to be remedied by a fuller analysis. For example, the somewhat implausible BY, OF, and WITH functions which we saw in the preposition-group rules become more acceptable when re-named AGENTIVE, PARTITIVE, and COMITATIVE; under these new names they may be linked to an analysis of the relation between nominals and their context, the 'deep' origins of cases (Fillmore 1969: 20 and passim), and cases. Nonetheless there remain functions whose motivation seems purely formal. For example, the Adjg rules capture the difference between {Ngfocus} and {Pgfocus} by functions NOM and PREP, though the difference between "yours" in:

"opposite yours"

and "to yours" in:

"adjacent to yours"

seems to be idiomatic and not semantic. This might warn us that the decision to regard these as Adjgs was wrong. It might imply that "opposite" and "adjacent to" must be treated as prepositions in constituents which we called {Complex}Adjgs. The reasons for not taking this view were given in section 5.4 of the last chapter.

There remains a group of functions whose status is particularly open to criticism. The functions are the place-markers which we called "buoys", and they include the Clause function POSTVERB and the Ng function HEAD. In a more complete grammar such functions are ubiquitous. As we have seen in earlier sections of this chapter, buoy functions are introduced to simplify the structure - building rules. Without them it would be an inelegant business to put functions together in their proper bundles, and it might sometimes be impossible. The trouble is that structure-building rules in their present form take only functions and function-bundles as their domain, so that buoys have to be functions too, despite their different motivation. The answer might lie in extending the domain of the rules to allow us to use buoys without having to pretend that they are ships. We should, however, be cautious in making such extensions to the scope of the rules, since the present criticisms were motivated by a desire for economy both in the absolute numbers of

feature and function symbols and in the scope of the rules.

The present grammar retains functions. The prime objective of the theory is to explain the phenomena of language in terms of what language is for, form in terms of function. A theory which codes formal and functional analyses in different symbols, and rule derives one from the other, has an immediately comprehensible structure. We deduce functions from structures by "interpreting configurations of features" (Hudson 1972 b: 9): the disputed question is whether we should express our deductions explicitly as a stage in the generative process, or whether we should leave them implicit, for the interested observer to work out for himself. That is, do we state explicitly that an item is a clause SUBJECT, or do we leave that fact implicit in the statement that it has the feature {Ng {Clause}---&c} ? It was felt that at this stage of development the explicit alternative was clearer.

The explicit alternative is easier to use in a computer program. We have already seen how the constructor procedures associate the semantic representation of the constituent with one of the functions it performs: for example, the representation of the game is associated with the ACTOR function in

"The game began ---"

This device seems to be somewhat more than a programming trick. It seems to be a rather plausible way of weaving together the semantic and syntactic representations of grammatical items. However, the alternative formulation would make this device impossible. The program would have to determine which constituent to link with the semantic representation by an examination of feature-sets and feature-dependencies, {Ng {Clause}---}. This examination represents the work now being done by the Feature-Realisation and Structure-Building rules; functions summarise the grammatical environment, so if the summary is not done in the grammar it must be done by a specialist procedure. The explicit alternative adopted in the present model is perhaps easier to comprehend, and is correspondingly easier to program. We are faced with a choice of formulations, one of which is more economical, the other of which is possibly clearer. Experience with the construction of larger grammars may incline one to economy, while an attempt to make a programmable model would emphasise the value of clarity.

Functions retain the advantage of clarity only so long as they retain their intuitive plausibility. They must identify a convincing syntactic or semantic job. For example, the word "although" in

"Although he's a fool, he looks sharp enough"

has the BINDER function, subordinating its clause to the major clause. Likewise the process participant functions are clear and helpful because they express an immediately comprehensible semantic analysis. The INITIATOR function in:

"The sergeant marched the prisoners"

must be the sergeant's. Of course the present system makes only those distinctions it needs to make. The participant roles of ACTOR, INITIATOR, and GOAL are adapted from Halliday (1967-8) because they are adequate for the program's needs. It would however be absurd to pretend that these roles alone present a complete analysis of the possible relations between a process and participants. We may hope that the distinctions we would need to add would correspond with 'deep' cases (Fillmore 1969) and with the participant roles identified by Schank (1970) and Bobrow (p.c. 1973) in his projected semantic network. For example, Halliday (1967 part 1: 55) proposed a 'benefactive' role within the transitivity system, instantiated in

"She gave John the bananas"

by "John", and in

"She knitted him some socks"

by "him". We know that if we move the 'benefactive' nominal after the direct object we shall need two different prepositions in the two cases:

"She gave the bananas to John"

"She knitted some socks for him"

and indeed Schank distinguishes "recipient" from "benefactive" roles. This is a comprehensible and natural distinction, for we are already well aware that whereas John received the bananas, he may never have received the socks. A distinction which must be made on grammatical grounds, to explain the use of "for" or "to", turns out to correspond with a distinction made by Schank for his analysis of events such as buying, selling, giving, exchanging,stealing and so on. The function notation in such cases is clear and helpful.

In a fuller grammar of English it become harder to give each function an intuitively plausible meaning apart from its purely formal motivation: we have already noticed some of the problems which arise. As this happens, the function notation ceases to be clear and our goal of giving a functional account of language become obscured. This problem needs a solution if the argument for retaining functions is to retain its force.

CHAPTER 6

6.1 Introduction

This chapter describes the program which produces English, attending in particular to the relation between the program and the grammar set out in the last two chapters. The aim is to see how certain rules can be applied to govern the conversion of stored information into English discourse: the program is an extremely precise statement of the rules and its logic will be carefully examined. However, nothing will be said which requires an understanding of program code, partly because it is of no intrinsic interest, partly because the programming language involved is not in widespread use and so would not be generally understood. The only knowledge of programming technique required in what follows is that a computer program can be divided into parts, which we call "procedures". A procedure has a special job to do, and can be told to do it by any other procedure which wants the job done. When invoked, a procedure may receive up-to-date details of the job ahead, and can report back on the outcome. As we mentioned in section 1 of chapter 5, the particular computer language used for this program call procedures "functions", but we shall continue to reserve that word for grammatical functions.

The last two chapters have set out the grammatical rules used by the program in producing English. The arrangement of the systems network generally corresponds

closely with the organisation of the program: that is when the program is assigning features to an item, it does so in order of increasing delicacy, from left to right through the network. However, where the network shows simultaneous choices, the vertical arrangement of systems has been chosen for clarity and to minimise line-crossing and not for its correspondence to the logic of the program. So far as the realisation and structure-building rules are concerned, the rules set out in the last chapter are a transcript of part of the program. The rules are exactly those which, by way of an interpreter, the program uses.

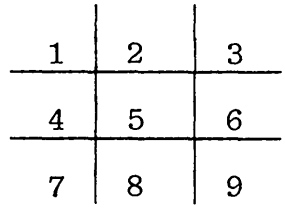
The {COORDINATION} systems are an exception. We have already seen what problems coordination presents to systemic grammar, and that the solutions offered in the present grammar are inadequate. The program does not traverse the {COORDINATION} systems as shown, and it does not employ realisation and building rules to govern the construction of a coordinate item. Instead, as we shall see later in this chapter, a procedure constructing an item checks if it is {Coordinate}, and, if it is, at once builds the appropriate immediate constituent structure.

6.2 An outline of the program

This section outlines the program, explaining briefly

what the program has to work on and how it is organised to produce English from this input.

The input to the program is a list of moves in a game of noughts-&-crosses. The moves are taken to have occurred in the order given and must be legal moves in that order, although they need not comprise a completed game. All the players have their own names, "Dan", "ACD", "Claribel", and so on. The squares, too, have names, as follows:



A move is specified by mentioning who the player was, and what square he took. Where the move came in the game is shown by where it comes in the list of moves.

The program accepts an arbitrary list of legal moves in this form, but it is also capbale of playing a game with an opponent and remembering what happened. In this case it adds to each of its own moves a note of what tactics it was employing at the time, to save re-computing this information when it comments on the game later. We shall come across 'mistakes made by the program' later in this chapter; these can arise, of course,

only when the program is presented with a jeu accompli, since the program's criterion of a mistake is that it would not have made that move itself, using its own planning procedures.

Given this list of moves, the program decides what moves are to be described in the first sentence and then describes them, making each move on a board in its memory as it describes it. It then returns to the list of moves, to design and construct the next sentence, repeating the cycle until it has described the last move.

The next section describes in detail how the program designs sentences: generally the two aims are to group moves which have a tactical coherence in the developing context of the game, and to keep the sentence length within a maximum of three main clauses. Mistakes are mentioned in the right circumstances. The design procedure is in two parts: the first performs a tactical assessment of a move in the momentary context, the second uses the first to decide how to group moves. The design for the sentence is expressed semantically, and the design procedures do not directly specify the syntactic structure of the sentence. Indirectly, however, they do, because the decisions of the designer procedure leave the construction procedures no freedom to vary the arrangement of the main clause of the sentence

The design procedure places the sentence design at the root of what will become the constituent structure tree of the sentence. The construction procedures then grow the tree until every branch terminates in a word. The root node is a sentence node, and the rest are clause, group, or word nodes. For each type of node there is a specialist procedure which grows the immediate descendant nodes. A construction procedure receives a specification of its task in the form of a semantic representation and a partial or complete feature list for the item to be constructed. If the feature list is complete, the constructor simply invokes the realisation and structure-building rules appropriate to that type of item. Usually, however, the feature list needs completion. Each constructor specialises in traversing the systems sub-net for its type of item: it knows how to select the right feature from each relevant system. In making these selections, the procedure may use syntactic informaiton, and semantic specialists which can report what game situation has been reached in the description so far, what has been mentioned recently, what other moves the current sentence will describe and so on. In particular, there is a range of specialist procedures which know about noughts-&-crosses and can explore alternative referring expressions for a particular square, line, or move until a satisfactory one is found.

When the tree for a particular sentence has been completed, the words at the leaves are printed, and the tree is reduced to a heap of logs ready for recycling into another tree. So the constructor procedures don't have access to the constituent structure of the last sentence when deciding upon a form of expression, but instead can consult an index of things recently mentioned. Winograd kept the entire analysis of the last sentence available to the current one, in order to guide the interpretation of pronouns in particular (1972 p 161): for example the "focus" of a preceding question is a priori a more plausible antecedent than anything else in that sentence. The current program does not have the sophistication of pronoun construction which Winograd gave their decoding, and that is a natural direction for development.

When the list of moves is exhausted, the programs checks whether the last move finished the game off: if it didn't, it comments. It then stops, having completed its task.

It will be remembered that messages such as

"I am going to start the game"

or

"The game has been drawn"

are produced by the program when playing a game. These messages are produced by exactly the same procedures as

produce game descriptions, except that the design stage is bypassed because the content of the messages is standard.

6.3 Designing a sentence

6.3.1 A detailed description of the design procedures.

Deciding how to express oneself is extremely difficult, and few have the skill of conveying novel material clearly. Often a speaker fudges over alterations of course in mid-sentence by the use of gestures or expressive intonation, so that a mere transcript of his talk may be very hard to follow. The present program has a somewhat easier task, because it is not interactive: like the club bore it takes no account of its hearer's actual reactions, but proceeds on the basis of some simple assumptions about its hearer. This section explains how it decides what to include in the next sentence, and we shall find that although it formalises only a few of the considerations borne in mind by the fluent speaker, the decision procedures are quite complicated.

The two principles which inform the design procedures are that each sentence should convey a coherent body of information, and that no sentence should be too long. A precise meaning, can be given to the first principle, because the designer is concerned simply with noughts-&-

crosses and assumes that the hearer shares the program's knowledge of the rules and interest in the progress of the game. The second principle is stylistic, and the meaning we give it is correspondingly arbitrary. The length of a sentence is restricted by the design procedures to three main clauses. A more interesting procedure would take account of the appearance of complex and lengthy referring expressions in the sentence as it was built, and would prune the planned content of the sentence accordingly. This would raise a number of difficult and interesting problems concerning the effect of anticipated sentence components upon preceding conjunctions and referring expressions. For example, the site of the single conjunction in a {Coordinate}, non {Multiconj} item such as the list "A, B, and C", is before the last coordinated constituent, so reducing the number of coordinated constituents alters where the conjunction goes. Or again, a restrictive relative may turn out to be so large when we come to construct it that it is best converted into a parenthetical sentence on its own, thus perhaps leaving a noun-group half finished:

> "...that wheel which - you know, it fell off the bus as we were waiting at the bottom of Leith Walk...".

A procedure which could alter the sentence design as the sentence was being built might naturally produce exactly those infelicities of which humans are constantly guilty.

This topic was raised at the end of chapter 1.

The design procedure has two parts. The first assesses individual moves when asked by the second to do so. It sees what the given move amounted to at that particular point in the game by putting itself in the place of the player, who made the move and then making gedanken moves according to various strategies until it finds a strategy which produces the given move. The point of this is to see what it would itself have been doing had it made the move, because this is what it will say the player was doing. As we have seen, if the move was one the program made in a game it has just played, the move is already tagged with the strategy used at the time: this saves re-computing the information, and provides a natural criterion by which the program could later decide whether to use the present tense with perfect aspect of an unfinished game which it remembers playing:

"---the game hasn't finished-"

or the simple past:

"--- the game didn't finish---"

The integration of the descriptive and game-playing procedures is meant to capture the fact that we do indeed ascribe to others our own limitations: the novice cannot intelligently describe the master's play. The program

in fact plays a less than optimal game, in order to avoid constant draws and so to widen the variety of games arising. Its descriptions therefore sometimes miss the point of a player's move: it may complain of a failure to pose a threat, not appreciating that the ground was being prepared for a fork. It would be a trivial matter to make the move assessment more knowledgeable, but this course has quite deliberately been avoided.

The assessment procedure runs through its strategies in an order which corresponds to their game interest and stops as soon as a matching move is generated. So if it found that a move was a "fork" and hence was also a "threat" and involved "taking", it would stop upon finding that the move was a fork. It thus always seeks the most significant aspect of a move for its assessment. In the context of noughts-&-crosses we can state an order of significance easily enough, and by assuming that the hearer is not a learner but knows the game as well as the program, we enable the design procedure simply to select the most significant assessment. In a more varied universe of discourse it might be hard to determine what aspects of a situation to mention. As Bobrow pointed out (p.c. 1973), Mary might be said to:

> "enter the shop, approach the soap counter, select a yellow bar, ---&c"

or "buy some soap at Boots"

or "go shopping".

This problem is particularly acute for systems which internally represent knowledge in a sophisticated system of primitives. They must respond at an appropriate level of reintegration unless they are to be merely comic. Few would see at once that a man who:

> "caused himself to cease being at A and to be at B at a later time by repeating a cycle in which the foot more distant from B was placed adjacent to, but beyond, the other in the direction of B" (Levin, p.c. 1973)

had walked.

The move-evaluation procedure checks an offensive move to see if it was also defensive; if it was, both aspects are mentioned. If the move was the first, or won or drew a finished game, the procedure notes the fact in its evaluation. Finally this procedure checks whether the current move was the only one which the selected strategy could have produced in the circumstances. If the strategy, for example "threatening", could have been carried out in either of two ways, the procedure adds a note that the particular square was "taken". Having made these checks, the procedure has done its job.

The second design procedure decides how many moves to describe in the next sentence, what conjunctions to use between move descriptions, and whether to mention any mistake it detects. Chapter 2 explained the principles of discourse planning which underly the procedure, and gave examples of the different groupings and conjunctions for which it is responsible. Here we see how it calculates relations of sequence and contrast between moves, how it decides to mention a mistake, and finally how the procedure is related to the {COORDINATION} systems.

The first design procedure assessed moves by their tactical significance, that is, by seeing what threats the move blocked and what threats it posed. The relation between a pair of move descriptions is contrastive if an expectation aroused by the first is disappointed in the second; if no particular expectation was aroused, or if the expectation was confirmed, the relation is sequential. Expectation can be calculated in terms of threats posed and baulked. If the first move makes no threat, no expectation is aroused, although the previously existing situation may, of course, arouse certain expectations. If the move makes a single threat, an expectation is aroused and may be disappointed by a parry. If the move makes two simultaneous threats, an expectation is aroused which cannot be disappointed on the next move - unless of course the opponent was

already in a position to win, and now does so. Fundamentally, then, the design procedure has a simple task: it must simply balance the offensive aspect, if any, of the first move against the defensive aspect, if any, of the second move.

First of all, then, the procedure must decide which aspects of the two successive moves will be juxtaposed and so determine the relation between the move descriptions. To design:

"...you blocked it and threatened me, but I blocked your diagonal"

the procedure calculated contrastive "but" from "threat...block" which are adjacent. In designing:

"...you threatened me by taking the corner, but I blocked your..."

the procedure made the same calculation, although the subordinate clause "by taking..." intervenes. The procedure must make its calculation before the clause structure of the sentence has been settled, but it takes account of the semantic factors which will influence that later calculation. Now the objective is to find the two aspects which will be mentioned in adjacent independent clauses; the method is to select the more general or significant aspect of each move-assessment where both aspects are offensive or both defensive, otherwise to select the offensive aspect of the first and the defensive aspect of the second. This pair of aspects is used to calculate not only the relation between the move descriptions and the conjunction which will express it, but also whether the current sentence must be terminated before the first

move, before the second move, or later.

6.3.2 On being surprised - sequence & contrast

Having selected the juxtaposed aspects, the procedure weighs them against each other, expectations aroused against expectations disappointed. We must now see how this elementary principle guides the sentence designer in the varying circumstances of the game, bearing in mind that the designer is trying to put related moves within a single sentence and to express relations by an explicit conjunction when possible.

Sequential conjunctions

If the first of the juxtaposed aspects is defensive or neutral, no expectation was aroused, and so, whatever the second aspect, a sequential conjunction is appropriate:

> "---I blocked that, and you took the middle square".
>
> "---you took the opposite corner, and I took another".

If the first aspect is offensive and the second is either neutral or insufficient to ward off the attack, the relation is sequential likewise. However, other considerations may affect the design. "Threaten" and "fork" are considered in turn.

If the first aspect is "threaten" and the response was a "block" or "win" the relation is contrastive and we are not concerned with it just yet. Alternatively the threat was not blocked; the relation is then sequential, but the second move was probably a mistake and a new sentence will be started to express it. But sometimes a mistake is not mentioned, as we shall see later, and then the sequential relation occurs, with or without an explicit conjunction:

"---you threatened me, I took the middle square, (and so you won---)"

The same applies in the version of the program which mentions vacuous threats, ones made when the opponent can ignore the threat and win on his move:

"---you threatened me, I forked you, (and so you won---)"

If the threatening first aspect was itself an illusory threat, the second aspect is likely not to be a block since the opponent should win without more ado: the relation between the two aspects in this case is considered in the section on mistakes.

The first aspect may be a "fork", and in this case the sentence is usually terminated after it. Normally a fork leads to a win, and the designer will group in a new sentence the hopeless defence and the inevitable ending, or, if the opponent put up no defence, his

mistake with its consequence.

In two cases a fork does not presage victory. The fork may be an illusory threat, followed at once by the opponent's win. This is considered later. Alternatively, the fork may not come to fruition, either because the game remained unfinished or because the player missed his chance. To avoid having the last move in a sentence on its own in the former case we have:

> "You forked me, and I blocked your diagonal (and threatened you. The game hasn't finished.)"

Otherwise the current sentence is ended after the fork.

Contrastive conjunctions

The relation between two aspects is contrastive if the second kills an expectation aroused by the first. We shall find that contrast differs from simple sequence both in that it may be expressed in different ways, and in that it may affect the way moves are grouped into sentences.

In the simplest case the contrastive relation is expressed by "but":

> "I threatened you by taking a corner, but you blocked my edge".

If the relation falls across a sentence boundary, "but" is best replaced by "however":

"...but you blocked my edge and threatened me.
However I blocked your line."

In chapter 2 we noted that "however" is suppressed if the sentence it introduces itself contains two tactics contrasted by "but", and saw why that is necessary.

A contrastive relation may be foreshadowed by "although" so "P, but q" may be expressed by "Although p, q". The point of this is to warn the hearer in advance that whatever p might lead him to expect will be confounded by q, so the program uses this alternative to describe a hopeless attempt to stave off defeat:

"Although you blocked one of my edges, I won by completing the other".

This use of 'although' entails subordinating the description of one move to the description of the next. The design procedure re-arranges the representation of the moves to signal the clause-building procedure to put the first in a subordinate clause, but does not otherwise trespass upon the territory of the clause construction specialist.

The design procedure arranges an 'although...' clause also in the case of some mistakes. When the program mentions a mistake it describes what the better move would have been, and the relation between the hypothetical and the actual is naturally contrastive. The modal verb

in the description of the hypothetical move warns the hearer that he is entering realms of speculation, but the use of an 'although...' clause gives him an earlier signal. The design procedure lacks a well-motivated criterion by which to decide whether to use a concessive clause as well as the modal verb, so it uses a simple stylistic test, namely whether the hypothetical move is to be described in one clause or two. If one, the hypothetical fits neatly into an 'although...' clause:

> "Although you could have forked me, you took the square opposite..."

If it has two aspects, the procedure leaves the contrast to be expressed by "but":

> "If you had taken a corner...you would have threatened me, but you took..."

It will be at once apparent that to express the second example in an 'although...' clause would be more clumsy:

> "Although if you had taken...you would have threatened me, you took..."

even if we take the trouble to put the main hypothetical clause, the apodosis, first:

> "Although you would have threatened...if you had taken..., you took..."

We often arrange an 'although...' clause after the main clause, in order to drown even before birth the expectations it might otherwise arouse:

"The candidate failed the exam although he had pinned a fiver to his answer."

The program never makes this inversion because of certain problems with keeping its record of the name straight, and because the motivation of the inversion is too subtle for such a simple universe of discourse.

Before leaving the subject of contrastive relations, we should notice a distinction which the design procedure makes between a mistake which was just a missed opportunity and a mistake which was immediately followed by the other player's win. A mistake which was a missed opportunity may still have posed a threat, as:

"---you could have forked me, but you threatened me by---"

and in this case the threat arouses expectations whose disappointment is marked by a contrastive conjunction:

"Although you could have forked me, you threatened me by taking the corner opposite the one you took first. However, I blocked your diagonal."

The other kind of mistake occurs when the game situation just before the mistake gave the winner his chance to win: for example, he was threatening his opponent. So after hearing of the opponent's mistake we expect to hear of the first player's win. This means that the relation between the mistaken tactic and the win is sequential not contrastive, even if the mistake

is apparently a threat:

> "I...threatened you. If you had blocked my line you would have threatened me, but you threatened me by taking the corner..., and so I won the game by completing my line".

This remains true even if the fact that the mistaken move was a mistake is not mentioned explicitly:

> "...you threatened me. I threatened you by taking the square opposite the one which I had just taken and so you won the game by completing your edge".

In an earlier version of the system the latter example would have read:

> "...you threatened me. Although I threatened you..., you won..."

but the present version makes no exception for mistakes which are glossed over, and designs better English in consequence.

The design procedure marks a contrastive relation by "but" or, at the beginning of a sentence, by "however" only if the hearer is likely to understand immediately what two pieces of information are being contrasted. As chapter 2 explained, the procedure avoids having more than one contrast word per sentence. Rather than 'p, but q, but r.' it designs 'p, but q. However, r.', and rather than 'However, p, but q.' it designs 'p, but q.'.

It also avoids 'p, and q, but r' on the grounds that, even in the simple world of nought-&-crosses, the hearer is momentarily uncertain whether r is contrasted with both of p and q, or just with q. However, it permits 'p, but q, and r' because the contrast is understood to arise between p and q, before r has been mentioned. These rules cause much of the variation in sentence length. The design procedure tries to group the material into sentences in such a way that, if there is a contrast, it can be explicitly marked by "however" or "but". In order to group the material like that, the procedure often cuts short a sentence before the maximum of three main tactics has been included; for example, instead of 'P, and q, but r' it prefers 'P.Q, but r'. In this example the procedure would revise the decision to put p and q together, linked by a relation of simple sequence, when it discovered the contrastive relation between q and r. Similarly, a decision to introduce a sentence by "however" is rescinded if the procedure finds a contrastive relation within the sentence. The procedure might have been written in such a way that decisions once made were never altered, but instead the designer did the best it could with each relation it found. This would have resulted in a more staccato style, and in the absence of "and" where it might have been expected:

"I took a corner, you threatened me. However, I blocked ---"

The design procedure is based upon the two principles of sentence brevity and avoidance of surprise. The universe of discourse is extremely simple, and the two principles have been given simple interpretations. However, the problems mentioned in this section are sufficiently complex to suggest that energetic, if subconscious, paddling is responsible for the swan-like grace of the fluent speaker.

6.3.3 Mentioning Mistakes

This section explains how the system decides to mention a mistake. First, it declines to mention its own errors at all, as we have already said. Then, provided the move was made by someone else, the program sees if it would itself have made that move in the circumstances. If it would not, it checks whether its own move would have been better. This makes sure that it doesn't upbraid a player for a move equivalent to its own. If the player had no threat to meet, the program requires that the better move would have posed a tougher threat, or won. If the player had a threat, or threats, to meet the program ought, to be consistent, to require that the better move stop the threat; a fork cannot be escaped, and it is arguable that the program ought not to carp at the player for failing to blunt at least one prong. However, on the principle that one keeps fighting to the end, the program does in fact call a move a mistake if the move did not do all it might have to stem an attack.

In this game situation a human player very often gambles on an attack of his own (rather absurdly in a game as simple as noughts-&-crosses), but the program's tactics are cautious and defensive, and it never suggests such a gambler's gambit. However, before the program mentions a failure to block a threat, it first looks ahead to see what happened next. If the game stopped unfinished upon the mistaken move, the mistake is mentioned. If the game continued, the question is whether the attacker took advantage of the current player's mistake. If he did, he won the game and the mistake is mentioned: in fact, if the win was possible only because of the mistake the program points this out by 'so':

> "If you had blocked my line, you would have threatened me, but you took the corner adjacent to the one which you took first, and so I won by completing my line".

If the opponent did not pluck the fruit of victory, that itself was a mistake. The game description becomes a little hard to follow if hypothetical alternatives accumulate, and since missing a win is a more interesting mistake than not stopping one, the choice of the program is to mention only the later mistake.

It will be clear that the program's treatment of mistakes is very simple. The motives for mentioning mistakes are various, but one motive is to educate the

hearer. If the present system could analyse a player's skills over a sequence of games, it would be possible to motivate the selection of mistakes for mention rather more satisfactorily than has been done here.

6.4 Introduction to the Construction Procedures

The remainder of this chapter explains how the constructor procedures work. As we have seen, there are specialist constructors for each type of grammatical item, and each constructor has three standard stages. First, the constructor examines the item's feature list to see if it is complete; if not, it completes it by reference to the semantic representation of the item and perhaps by other criteria too. This stage is what the rest of this chapter is about. Second, the constructor uses the rules given in the last chapter to specify the immediate constituent structure of the item. This stage has already been fully explained. Third, the constructor invokes the specialist constructor appropriate to each immediate constituent. It discovers which one to invoke by examining the partial feature list which realises each constituent's function-bundle: for example, the (SUBJECT ---) is realised in {Ng Subject} , so the clause constructor knows that the Ng constructor specialist is needed for this constituent.

The first stage of each construction procedure is to complete the item's feature-set by traversing the remainder

of the appropriate systems sub-network. To decide whether the item should be given a particular feature normally involves the invocation of other specialist procedures to examine the syntactic or semantic context. Sometimes these specialists construct bits of English as they resolve particular points. For example, in order to see whether a noun-group is {Modified} , {Qualified} , or neither, the program must first determine whether any restrictive element is necessary and then, if so, whether a modifier can be found to do the job:

"---the adjacent edge"

or whether the restrictions has to be expressed in a qualifier:

"---the edge adjacent to the corner which you had just taken".

To resolve this, the procedure must try actually to construct the noun-group constituents required, modifier or qualifier. This means that the constituent is in being before the relevant feature is added to the item's feature list. Consequently when the cycle of Ftr, SB, and FNr rules has run its course, the third stage of the noun-group construction procedure does not call upon a further procedure to make the constituent, but simply places in position the one already made. It will therefore be obvious that the program's operations cannot be categorised as working "top-down". It does not invariably construct an item by determining its feature-set, thence determining constituent-structure, and finally building

each constituent. Instead, syntax and semantics are woven together and dependent on each other, either one being able to take control as the situation demands.

Some of the merits of a systemic functional grammar as the basis of a productive system are thus clear. It analyses in the systems network the factors which influence structures, and defines when in the productive process, and how, these factors are to be taken into account. If the evaluation of such a factor requires the production of a constituent there is no question of placing it at some temporary site in constituent structure and subsequently moving it by a structural transformation: the only constituent structure in the theory is surface structure. But the grammar provides a natural solution to the problem of where to keep the constituent until its right and final site is known. The constituent is simply associated with a function which it will ultimately perform, for example MODIFIER or QUALIFIER. Then when this function realises the feature-set and has been placed in its ordered bundle, the constituent can readily be placed in position. The grammar thus relieves the program of any need to revise provisional constituent-structure decisions by structural transformations. Decisions are made only when they can be final.

6.5 Making Clauses

Under the overall control of an administrative monitor, every clause is fabricated by the specialist clause constructor from a blueprint supplied by the design procedures. The blueprint of a clause is a crude semantic representation of what the clause must convey. We shall examine such a representation in order to see what the clause constructor works from, but the representation will be trimmed of certain superfluities which are of use only to the trouble-shooting programmer. At its simplest the representation comprises a list of three variables, giving respectively the Initiator, the Process, and the Goal, thus

{ < Initiator > < Process > < Goal>}

For example, the representation of the clause

"I threatened you"

where "I" is the program Proteus, and "you" is the player Dan, would be

{ < Proteus > "threaten" < Dan>}

The Process element is a quoted word, as shown, but is simultaneously the internal name of the tactical procedure used in making, or recognising, a "threatening" move. The Initiator and Goal elements are rather complex internal records whose details need not concern us once we have noted that they include the quoted word "Proteus" and "Dan".

The clause constructor must first complete the feature-list of the clause item. In the example we are considering the feature list starts empty; no features are given to the constructor. Later in this section we shall revert to a more detailed consideration of features which the clause constructor may be given. But at this point the question is why no features are supplied: it might, for example, be expected that at least {Clause} and perhaps {Indicative} would be given, as realising a "statement" function being performed by the clause item. The reason they are not given is simply that the scope of the grammar does not extend beyond the independent clause: the grammar does not formalise any sentential or conversational functions to be realised in features of independent clause items. There is thus no way of supplying these features to the clause constructor consistently with the operation of the rest of the grammar. Two alternatives were available in this situation: the features might be supplied by the administrative monitor, or they might be generated within the clause constructor. The solution adopted was a compromise. Features are given to the constructor by administrative procedures only exceptionally, under circumstances explained later in this section. Normally the constructor generates for itself all features of independent clauses. Dependent clauses are, of course, a different matter since they perform functions formalised in the grammar, and the realisations of those functions constitute a

a partial feature-set supplied to the constructor function.

The clause constructor, then, has an empty feature list and the representation {<Proteus> "threaten" <Dan>}. It can at once add {Clause} to the feature list, and then {Simple} by inspection of the representation. The {MOOD} systems are traversed by exception; unless told otherwise by an external procedure, the constructor makes all independent clause {Indep}, {Indic}, {Declar}, and {Past}. The {ADJUNCT} systems are not entered in this case, because the representation contains no adjunct item, and features from the {ASPECT} and {UNDERSTANDING} systems are always supplied by exception from outside so the constructor can ignore them here. There remains the {TRANSITIVITY} subnet. The constructor looks up 'threaten' in a lexicon and thence determines the features {Ext Eff}. As the Initiator is given in the representation, the unmarked choice of operative, {Op}, is made. Finally the clause is {Gtrans}. The choice of this last feature ought to be made by reference to whether the Goal element is present in the representation; it should, in other words, be semantically determined. However the present constructor first sees whether the verb is such as to occur in a {Gintrans} clause as "You won", as well as in a {Gtrans} clause, as "You won the game", and then makes an arbitrary choice if a choice is open.

If the verb had been different the clause might

have been {Descr}, in which case an arbitrary choice of {Op} :

"You began the game"

or {Middle} :

"The game began"

is made, since the procedure's only criterion for theme selection is that the passive voice in a {Receptive} clause is the marked alternative to the active. The implicit assumption which legitimises this arbitrary choice is that the Initiator process-participant omitted from a {---Descr Middle} clause is not entirely lost to view, but will reappear in an adjunct as a determiner:

"---with your taking a corner".

If this assumption could not be made, the procedure could not thus arbitrarily leave out information. A more adequate analysis of theme, requiring a slightly wider scope for the discourse than that open to the present system, would be a natural development of this part of the program. A similar limitation accounts for the arbitrary choice of {Gtrans} or{Gintrans} which was mentioned earlier.

The clause constructor may be given some features at least, rather than the empty list of the last example. These features are of two sorts, those which are ready to hand at some point before the clause

constructor starts making the clause they characterise but which could be calculated only with greater difficulty by the constructor itself, and, on the other hand, those which realise functions of the grammar in the regular way.

The first sort, "anticipated" features, may be calculated either by an administrative procedure which calls for an {Independent} clause in the course of playing or describing a game, or by a constructor procedure making an item of which a {Dependent} clause is part, or by a semantic specialist calling for a relative clause to be made. Each of these cases is now examined.

When invited to play a game, the program may ask

"Will you start the game?"

The constructor making this clause is supplied with the feature {Interrog} by the procedure which organises the play. Likewise, the procedure which organises the game description supplies {Negativ} when requesting the final comment

"The game hasn't yet finished"

The semantic representation of the clause is too elementary to lend itself to conveying negativity in any interesting way. In any case the example has present rather than past tense, and it is arguable that it is an administrative comment rather than part of the game description. As such it might be distinguished

from

"The game didn't finish"

and its negativity would then stem with quite proper directness from the administrative procedure rather than from the representation of the game. The present system may therefore be right in having the administrator tell the constructor to make the clause negative, even if the instruction should appear within the semantic representation rather than as a feature.

Both constructor procedures requisitioning dependent clauses, and administrative procedures may call for a clause with a particular tense, aspect, or modal auxiliary. They do so by way of features. The clause constructor could calculate these features, but the procedure requiring the clause usually knows from its own context which features are wanted and so saves time by noting them in anticipation.

A special situation arises in the case of a coordinated clause which understands its subject:

"---(I blocked it and) forked you",

Such a clause is {Ustand Ustandi Ustands} . It would be very wasteful for the clause constructor to check every clause for these features, since a complex investigation of the clause environment would be required. So the constructor of the parent {Coord} clause

makes itself responsible for noting these features, which it can do after a simple calculation.

The third source of "anticipated" features is the semantic specialists which may call for a relative clause to be made. From what was said in the last section it will be apparent that this may occur in the course of trying to construct a noun-group and so before the cycle of Ftr, SB, and FNr rules have applied to the Ng's feature-set. Therefore the semantic specialist has to give the clause constructor the features realising (RCLAUSE), namely {Clause Dependent Finite Relative}. In addition it saves the constructor checking which process participant is relativised, by noting the appropriate features from systems 21,22,26, and 300-302. Finally the semantic specialist may decide that a particular time adverb or aspect is necessary. If so, it marks its requirements by adding features from systems 31,40,41, and 42.

The other class of features which the clause constructor may be given comprises those features of {Dependent}clauses which realise functions, according to the rules given in the last chapter. For example, an "although---" clause, as

"Although you could have forked me, you took ---" performs the functions (ENVIR = CLAUSE = CONCESS) in

the superordinate clause, and these functions are realised in the feature-set:

{Clause Dependent Finite Bound Thobound} .

So when the clause constructor makes an "although" clause, it finds these features already present. Mutatis mutandis, the same is true of conditional clauses.

When considering the example{< Proteus > "threaten" < Dan >}, the question of co-ordination was put on one side. A word must now be said about the treatment of co-ordination and adjuncts. The representation:

{ <Proteus > "threaten" < Dan > "take" < square 5 >}

is the blueprint equally for:

"I took the middle of the board and threatened you"

and for:

"I threatened you by taking the middle of the board".

The clause constructor decides which form to build, whether

{Coord Linked Conjunc Binary}

or

{Appadj Methodadj}

by considering the context. If the current clause constitutes the whole of the current sentence, the former alternative is selected. If the clause is coordinated with others, the latter is preferred. The

decision is thus taken on stylistic grounds, to avoid excessively long chains of co-ordinated clauses. We should notice that the clause constructor takes this decision only so far as concerns the clause structure of the description of a single move. Decisions about the clause structure of a sentence describing several moves are taken by the sentence designer. So in

> "I took the middle of the board and threatened you."

the conjunction "and" and the coordinate structure was selected by the clause constructor in the way just described, whereas in

> "You began the game by taking a corner, I took one of the adjacent ones, and you took the other."

the three coordinated clauses were specified by the designer. This division of responsibility is intended to capture the fact that designing a sentence is different from designing a clause. Different factors have to be weighed in the two cases, for the context of a sentence is the discourse, while that of a clause is at most a sentence. When sentence and clause coincide there may seem to be a clash of responsibility, but the grounds of decision are so different in the two procedures that no confusion should arise.

If the clause constructor decides to make the current clause {Coord---}, it at once constructs the

needed immediate constituents, invoking the specialist constructor appropriate to each. It passes to each {Simple---} clause constituent the feature set, if any, supplied to the coordinate item, less, of course, {Coord} itself. It may also mark the second coordinated clause as {Ustand--} in the way noted earlier, so that it understands its surface subject.

This section might be summarised by saying that the clause constructor may be given features which realise functions of {Dependent} clauses, it may be given features which do not realise any function but which are "anticipated" by some other procedure, and it must calculate the remaining features from the semantic representation and the syntactic context. The "anticipated" features are anomalous. Some would be eliminated if the scope of the grammar extended to include sentences and the functions of their constituents. The rest would be eliminated if the semantic representation of the clause were more sophisticated and so allowed the clause constructor to deduce these features as it does the {TRANSITIVITY} features, for example.

6.6 Making verb-groups

A study of the Verb-group systems in section 5.1 of chapter 4 reveals that every feature is the realisation of some function. The only exceptions are person and

number, systems 52,53. The finite verb agrees in person and number with its subject, so the correct choice in these systems depends upon knowing what the subject is. It is considerably simpler for the clause constructor to note these features for the future use of the verb constructor, than for the verb constructor to do so, and so that is what happens. Consequently the verb constructor does not have to select any features for the item's feature-set. We have already seen that the {Vg} item has no constituent structure, so the verb constructor simply has to use the feature-set to pick the right form of the verb. If the verb is lexical, the clause constructor associated the Process element of the semantic representation with the Vg constituent before invoking the verb constructor. If the verb is grammatical, the features of the Vg determine what closed - class dictionary verb to pick.

6.7. Noun-Groups

6.7.1 Overview

This section explains in some detail how the noun-group constructor completes the feature-set of the item it is making. It has already been hinted that this constructor places great reliance upon semantic specialists, but no attempt will be made here to explain how these specialists work, either individually or in harness as a team, except so far as is necessary to show how features are selected.

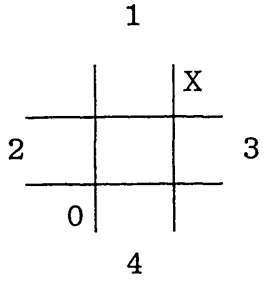
The objective of the entire system is to convey information as needed but with minimum redundancy, and the objective of the noun-group constructor is the same within its more limited context. The constructor is told what the referent is, and tries to find a form of words which denotes it. First the constructor tries to do the job by using a simple pronoun, a proper noun, or a common noun (or equivalently, the prostring "one") with an article. If this attempt fails, the constructor must add some restrictive information to the common noun, to confine its denotation adequately: we come to the criterion of adequacy in a moment. The restriction may be in any of the forms illustrated in the program's output, including a modifier, a qualifier of various sorts, a possessive determiner, or a possessive pronoun. The job of finding a restriction is dumped by the constructor on the lap of a specialist procedure, which at once takes over the enterprise completely. It tries to find a restriction which is adequate and at the same time as short and structurally simple as possible. Only when its search is finished does it return control to the constructor. The constructor makes sure that the feature set is complete, and thus ends its first stage.

6.7.2 The criterion of adequacy

The touchstone upon which both the constructor and the other specialists try their answers is a simple one.

A noun-group adequately denotes its referent if it denotes the members of the equivalence class of the referent, and nothing else. We leave till later the problem which arises when the noun-group denotes only a subset of the referent's equivalence class. The equivalence-class of the referent, which we shall abbreviate to its computer code name XEQCL, is the set of all entities which, in the momentary context, are equivalent for the purposes of speaker and hearer. In the little universe of discourse we are concerned with, this notion is given a precise meaning as follows.

The universe includes the players, the game and the board. Each of these entities is unique, and its equivalence-class has just one member, the entity. The remaining referable entities in the universe are parts of the board, edges, diagonals, lines, and squares. We may recall that the program gives a game-commentary which takes no account of the orientation of the board, but only of the tactical situation. So the equivalence class of a part of the board is the set of those parts of the board which are tactically equivalent to it in the momentary game situation. For example, in the situation:



the two empty corner squares are tactically equivalent, the edges marked 1 and 3 are equivalent, and the edges marked 2 and 4 are equivalent. Note 1 at the end of the chapter explains the calculation of the equivalence class of a board part, but the details need not concern us now.

The notion of the XEQCL captures an important aspect of the way we construct referring expression in English utterances. The little world of noughts-&-crosses is simple enough to make the calculation of an entity's XEQCL a straightforward matter: it would be more difficult if the present simple and static assumptions about the hearer had to be made complex and dynamic, as, for example, they would have to be if the hearer could ask the program for clarification of aspects of the commentary. Nonetheless, even the present simple calculation offers a starting point for such development.

6.7.3 Finding an adequate referring-expression

This section explains in rather more detail how the noun-group constructor selects features from the noun-group systems. What the constructor has to go on is a semantic representation of the referent, as:

{ < Dan > }

and a feature list with realisations of the noun-group's

functions. It will become clear that the criterion of adequacy just explained is the arbiter of the constructor's efforts, particularly when the constructor calls in the specialist restriction-finder.

The noun-group may be a clause, as

'completing my edge'

or

'my taking a corner'.

The constructor detects a {Clause} Ng by its appearance in the semantic representation of the referent, and it distinguishes the non-determined first example from the determined second one by seeing whether the clause is marked {Ustand...} or not.

The second important question for the constructor is whether the Ng is coordinate. Of course, {Clause} Ngs can be coordinate but as they happen never to be so in the program's output, computational economy dictated that the question of coordination be left to second place. The constructor makes the Ng coordinate if the entities in the semantic representation do not belong to a common equivalence class. Otherwise it will make a simple, plural noun-group. If the noun-group is {Coord}, the constructor at once recursively arranges for the construction of each coordinated constituent.

From now on we assume that the constructor is making

a simple {Nominal} noun-group. It proceeds at once to determine from the semantic representation the person and number of the referent and whether it is animate. If the constructor were constructing 'you' in:

"I threatened you"

from the representation {<Dan>}, it would at this point have the partial feature list

{Ng Object Nominal Sg 2 An}

The procedure now invokes a specialist to decide whether the noun-group should be represented by a pronoun. As was said in chapter 4 section 2, third person proref pronouns admit no modifier or other restriction. So such a proref pronoun can be used only when the present referent is identical with that of the antecedent. The first and second person pronouns in the singular denote unique interlocutors, and thus make unnecessary any reference to antecedents or equivalence classes, but third person pronouns pose more difficult problems. The pronoun specialist is considered more fully in section 4.4.6 of this chapter. In the example we are considering, the pronoun specialist would be responsible for adding {Pronoun Proref} to the feature list, and the only outstanding feature to be evaluated is definiteness, which we come to a little later.

In a different example, the noun-group might have been not a proref pronoun but a proper noun. The only proper nouns in the system are the names of the players. Each player has a name, and each name denotes a unique player. Then if the referent is a player and is not to be denoted by a pronoun, there is some proper noun which adequately denotes him or her. If neither a proref pronoun nor a name adequately denotes the referent, the constructor selects a common noun. The common noun may have to be represented by a prostring pronoun, as:

"the other one"

or to be omitted altogether, as in:

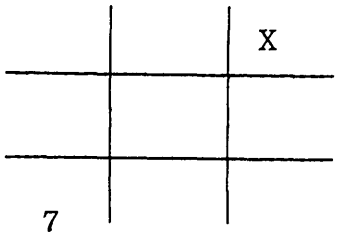
"the other"

but neither of these eventualities affects the calculation now to be described.

Every common noun known to the system denotes a set of entities, so the procedure selects that noun which denotes the smallest set of which the referent is a member. For example, a corner square is both a square and a corner, but the procedure would select "corner" to denote it, as denoting a smaller set than "square". This selection criterion violates the fundamental principle of economy of information: if the referent is adequately denoted by "square" it is extravagant to reveal that it is a "corner". True though this criticism is in principle, in practice the present procedure gives little away. Ordinary English is not well adapted to giving parts

of a three-by-three matrix a rotationally-invariant description, and in an earlier version of the system it was found that giving the procedure the chance to try the more general word first led to much the same referring-expression as at present, but with greatly increased expenditure of computation. In other words, for very many of the referents which occur in the range of game situations, the system's resources admit of only one adequate referring-expression and that one demands the more specific noun.

Every common noun denotes, within the present universe of discourse, a closed set of entities, and the constructor procedure must make a referring expression which denotes a subset of XEQCL. For example, suppose the referent were a corner square and the game hasn't yet started. The XEQCL of the referent is the set of all the four corners. "Corner" refers to the set of corners, and so a subset of XEQCL. This means that no restriction need be added to "corner", and indeed "a corner" is an adequate referring expression for a corner when the board is empty. Once the game had started, the procedure would have to find a restriction to select some subset of the corner squareswhich was a subset of the referent's XEQCL at that stage. Thus to refer to square 7 in the situation



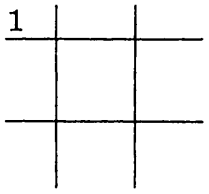
a restiction must be found which selects a subset of XEQCL, {7}, out of the set of corners, {1 3 7 9}, and so we have "the corner opposite the one X had just taken", or, in fact, "the opposite one".

The constructor procedure, then, selects the common noun as described, tests whether a restriction must be added, and if so invokes the restriction-finding specialist. This specialist is actually a team of specialists, whose points of interest are explained in section 7.7. Here we observe simply that the constructor procedure receives back from the team both the noun-group features appropriate to the restriction selected and also the restriction constituent itself. To continue with the last example, the noun-group constructor would invoke the restriction-finding specialist to help with square 7, and would be given both {Modified} and the {Simplex} Adjg terminating in the one word "opposite". In due course the constructor arranges the noun-group into:

"...the opposite one".

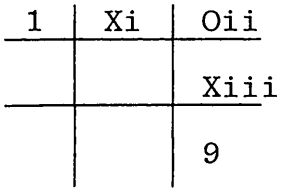
After the restriction-finding specialist has reported, the constructor enters system 64, to select {Def} or {Indef}. The constructor procedure, perhaps relying

upon the restriction-finder, regards its task as adequately discharged if the referring-expression denotes a subset of XEQCL. The referring-expression is definite if the cardinality of this subset is the cardinality of the referent set, otherwise indefinite. To revert to an earlier example, when we use "a corner" to refer to, say, square 1 before the game starts,



we use the indefinite article because the cardinality of the referent set,{ 1 } is not that of the set {1 3 7 9} denoted by the referring expression.

This formulation has certain advantages. It takes "overdetermination" into account, when the subset of XEQCL denoted by the referring expression is a proper subset rather than coincident with XEQCL. Suppose the board situation to be:



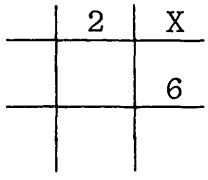
where Roman numerals indicate the order of moves. Suppose that the referent is square 1. Clearly XEQCL is {square 1, square 9} . The restriction-finder might eventually decide to "overdetermine" the expression, by mentioned the edges at whose intersection the

square lies:

> "the corner common to the edge of which X took the middle first and the edge opposite the square X had just taken".

This expression denotes the subset {square 1} of the XEQCL { square 1, square 9} , and the cardinality of the subset is 1. Since the referent set is also of cardinality 1 the expression is definite.

The same line of thought explains why we refer to square 2 in the situation:



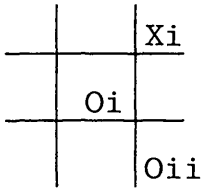
as "the middle of an adjacent edge". The referent set is {square 2} the XEQCL is {square 2, square 6}. The constructor invokes the restriction-finder, which proposes the qualifier "of an adjacent edge": this noun-group is, of course, indefinite, but as a restrictive qualifier it restricts the denotation of the ensemble to just that "middle" which falls within whatever adjacent edge we pick. But each edge has only one middle square. Therefore the cardinality of the set denoted is the same as that of the referent set, and the referring expression as a whole is definite as shown. What the restriction-finder actually does, having failed to find any other restriction, is to requisition a noun-group denoting the edge within

which the referent falls, "an adjacent edge": then it intersects XEQCL with the set of squares comprising the edge and so obtains the set {2} denoted by the whole expression.

We saw in chapter 2 that, as one can take only squares which are empty, the restriction "empty" or "free" may be left implicit. When this is combined with the mode of expression being reviewed here we could have expressions such as:

"(I took) the end of an edge adjacent to the square X took first"

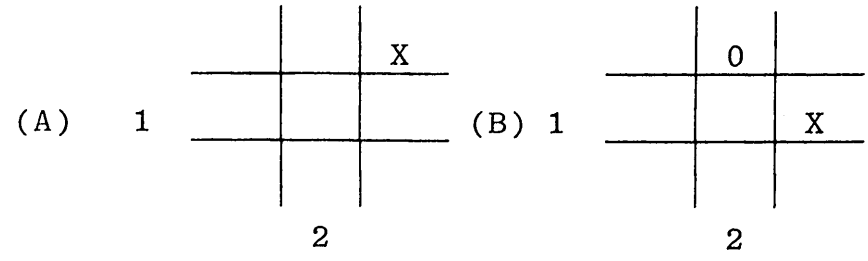
to denote square 9 in the situation:



despite the fact that every edge has not one but two ends.

6.7.5 Coordination and the referent equivalence class

If the referent comprises a set of entities, it may, of course, happen that they do not all belong to a common equivalence class.



In case (A) sides 1 and 2 belong to a common equivalence class,

"the edges opposite the corner X had just taken"

but in case (B) they don't. In case (B) the noun-group constructor observes that the equivalence class of side 1, comprising {side 1}, does not include side 2: it follows that a referring expression which adequately denotes side 1 will not denote side 2. So this is the cue for the constructor to make two coordinated referring expressions, the referent of one being side 1, and of the other side 2.

"the edge opposite the square X had just taken
and the edge opposite the one Y had just taken".

6.7.6 Proref & Prostring Pronouns

The noun-group constructor can use pronouns, and its problem is to decide when it should do so. Sometimes there is no choice. The relativised noun-group at the front of a relative clause must be a relative pronoun:

"---the square which you took first".

If the referent is the speaker or the hearer, a proper-noun or definite description is unacceptable except in the vocative case, so the procedure must use respectively a first or second person pronoun. These then, are straightforward cases.

Rather more challenging is the problem of deciding when a third-person proref pronoun should be used. It is worth remembering that these pronouns, both unmarked ("he, she, it,---") and {Deictic} ("this, that,---"), are more than a source of stylistic variety and more than an economy measure, used to avoid repetition of a longer referring-expression. Both types pick up the referent of their antecedent noun-group, and they therefore both have a deictic component, as we saw in Chapter 2. So these pronouns bind the discourse into a flowing and comprehensible whole: if they are not used when they might be, the speaker is assumed to be making a marked choice, to indicate non-identity of referents.

However the construction-procedure cannot commission a pronoun unless its antecedent will be apparent to the hearer, and an assessment of this question depends to some extent upon an assessment of the hearer himself. Hearers differ in attentiveness, intelligence, and knowledge of the subject, and although a clear speaker takes some account of these factors, he is not always

certain that his anaphora hooked onto the antecedent's referent in the hearer's mind. In many cases, then, the question whether a pronoun should be used can be answered only by "probably", rather than "yes" or "no". Winograd (1972 p 158), modelling the hearer, assigned to each potential antecedent a probability calculated from syntactic factors, later to be reviewed according to semantic requirements.

Syntactic criteria can certainly be constructed. First and most obviously, the antecedent is likely to be the most recent noun-group of the corresponding person, gender, and number, so:

"Bill saw a boat and he bought it"

is perfectly clear. However, simple proximity is a very weak criterion, and an example like:

"Bill and Fred met, and he said ---"

shows that uncertainty about the antecedent intended is not resolved by proximity.

"Bill gave the cat a mouse, and it died of fright" is also ambiguous, although "mouse" is closerto "it" than "cat" is. Another criterion is prominence: the more prominent the antecedent, the more tolerable is a "distractor" intervening between antecedent and pronoun. A noun-group is prominent if, for example it performs the MOOD-FOCUS or SUBJECT roles, or if it is the theme of its clause. If the antecedent of "it"

in the last example is made the theme, it becomes more prominent, so:

"The cat was given a mouse by Bill and it died of fright"

is probably the cat's, not the mouse's, obituary.

Langacker (1969) formulates some persuasive criteria in terms of constituent structure. He suggests that we define a 'command' relation between two constituent structure nodes, whereby if A precedes B in linear order, and neither A nor B dominates the other, and the S node which most immediately dominates B also dominates A, then A commands B. He then proposes to define certain 'primacy' relations, which include 'command' as defined and 'precede in linear order'. He finally suggests the negative rule that, given two noun-phrase nodes, NP^1 cannot be the antecedent for NP^2 if NP^2 has the two primacy relations mentioned. This rule is undoubtedly helpful, but, being negative, can only stop us going wrong rather than actively put us right.

But when these, and perhaps other, syntactic criteria have been used, their calculations may be puffed aside by the semantic interpreter. Examples come to mind too readily to need repetition here. Unfortunately we cannot rely upon semantic considerations to validate the use of a pronoun. Experience with the present program

has shown that English discourse cannot be constructed as though it were a crossword-puzzle: it may be undeniable that in noughts-&crosses something "blocked" must be a line but the following remains in effect incomprehensible:

* "The game started with my taking a corner, and you took an adjacent one. I threatened you by taking the middle of the edge opposite that and adjacent to the square which I had just taken, but you blocked it and threatened me."

Or again, the procedure cannot rely upon the hearer to realise that, in the present universe of discourse, only "games" can "finish": so it cannot conclude a lengthy game - description by "It hasn't yet finished" regardless of the distance back to the initial "The game began---"

The constructor procedure relies for guidance upon a pronoun specialist. This specialist mantains and consults two stacks of information, which we shall refer to by their computer code mnemomics as PREVREF and CANLIST.

PREVREF (the PREVious REFerents' list) is a stack of entities for which referring expressions have successfully been completed. Every time such an expression is completed by the constructor, the pronoun specialist adds to PREVREF a note of the noun-group's referent,

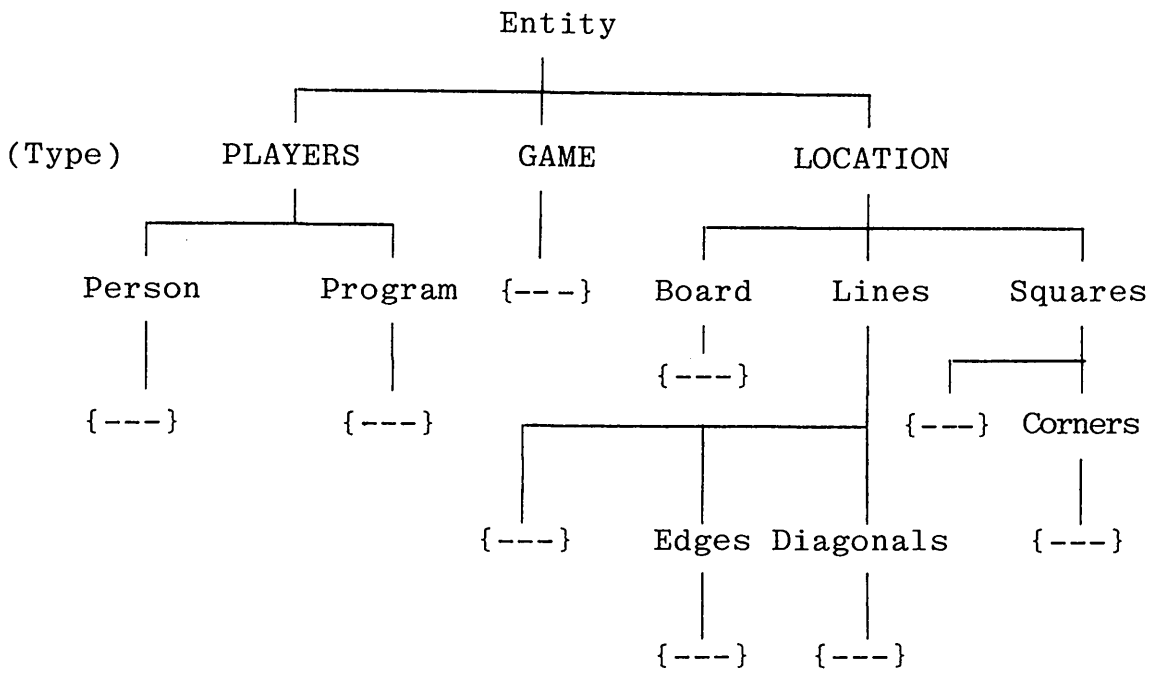
the noun used to head the noun-group, or, if a pronoun, the noun used to head the antecedent, and thirdly a copy of the noun-group's feature-set. Having added this entry, the specialist weeds out any previous conflicting entry. A previous entry conflicts with the current one if it has the same referent, the same noun, or the same person number,and gender as that one.The revised PREVREF then constitutes a pool of authorised antecedents and their referents upon which the constructor of the next noun-group may draw.

The weeding-out criterion as stated is too strict, since it would preclude the use of "it" in:

"You began the game by taking a corner, and I took the opposite one. It hasn't yet ended".

We observe that "a corner" shares person, number, and gender with "the game", and so would delete it when added to PREVREF. Some attempt must be made to capture the fact that a game and a corner are semantically not very confusable, so that anaphora to one over a distractor comprising the other will probably be acceptable. The pronoun specialist therefore exempts from deletion a conflicting entry whose referent is of a type which does not conflict with that of the current entry. The type of a referent is defined by reference to a simple tree of the entities of the little universe of discourse: in the diagram here {---} shows the set of entities

denoted and the three 'types' of entity are written in capitals.

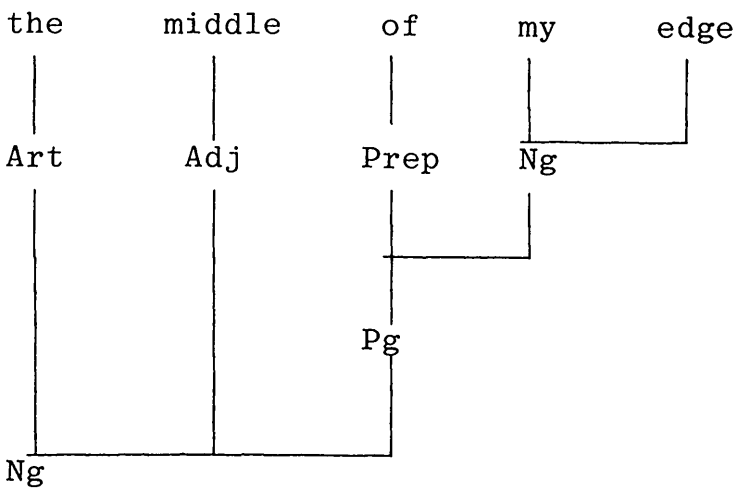


So the program and any person are of the same type, PLAYERS, but a corner and the game are not of the same type.

The weeding-out criterion needs to be further relaxed, since in its present form it would make impossible the use of "it" in:

"You took the middle of my edge and blocked it"

though the sentence seems quite acceptable. The constituent structure of the antecedent shows why the proposed criterion would prevent the anaphora:



The noun-group "my edge" is completed before "the middle of my edge", and so the former precedes the latter in PREVREF: the latter's arrival would then delete the former from PREVREF since the two noun-groups coincide in person, number, and gender, and in type. The reason for adding the noun-groups to PREVREF in this order, rather than in order of occurrence in surface structure, is to capture the fact that the whole noun-group denotes a square, not an edge, that the superordinate noun-group is a somewhat more likely antecedent for a subsequent pronoun than is the subordinate "my edge". We notice that even if the order of entry to PREVREF is amended to correspond with surface structure order, the weeding-out criterion remains too restrictive: the sentence:

"I took the free end of the edge you had just

taken the middle of and blocked it"

then raises exactly the same difficulty, since "the middle of" obliterates from PREVREF the entry for "the edge...".

It seems clear that one reason why "it" is acceptable in the example is that "block" tells the hearer that "it" must denote a line. The pronoun specialist might then leave conflicting entries in PREVREF, and authorise anaphora to these antecedents if the main verb imposed adequate semantic constraints upon the process participant in question. It would be possible to formalise these constraints by reference to the tree of entities set out on page before last, and we shall see later that the restriction-finding specialist does indeed do something of the sort in a different context. Nonetheless the pronoun specialist does not follow this course because it turns out that semantic direction of anaphora is ineffective in the present universe of discourse and results in intolerable obscurities. The point was mentioned earlier in this subsection and illustrated with an example.

Another reason why "the middle..." does not prevent anaphora to its constituent "edge" seems to be that "middle", like "end", does not have the status of a noun. The program in fact regards both words as adjectives. We observe that in consequence "the middle of an edge",

though obviously a square, is a less satisfactory antecedent for a proref pronoun than "the middle square of an edge " is. The anaphora in:

> "You took the middle of an edge, and I took the square opposite it "

is less immediately comprehensible than in

> "You took the middle square of an edge, and I took the one (square) opposite it".

As we would expect, such a noun-group offers no antecedent for a prostring pronoun: so

> * "You took the middle of an edge, and I took the one opposite it".

doesn't work. The pronoun specialist therefore regards a noun-group whose head is an adjective as being of lower status, unable to expunge from PREVREF a conflicting entry whose head is a nominal. Under these conditions both entries remain in PREVREF as potential antecedents, for proref pronouns.

The pronoun specialist authorises a pronoun only if its antecedent occurred in the present or in the preceding sentence, and so it deletes from PREVREF any entry whose antiquity has destroyed its usefulness. This rule is of course arbitrary, but produces sensible results because the break between sentences is normally a strong one: anaphora cannot easily span more than one such gap.

The CANLIST (Currently Active Node List) is a list of the referents of noun-groups currently being constructed but not yet complete. For example, during the construction of "your edge" in:

"the middle of your edge"

CANLIST would contain two referents, a square and a line. This stack is used incidentally to stop the restriction-finding specialist referring to a part of the board by relation to itself:

* "the corner which is opposite the one opposite it"

but its primary purpose is to supplement PREVREF by noting potential antecedents before their completion. The antecedent of a relative pronoun is not complete when the pronoun is constructed

"the man who sat on my hat"

and we must also be able to cope with

"The man who physicks himself (has a fool for a patient)"

in which the referent of "himself" can be identified only with the aid of the relative clause in which it occurs. Examples of the latter type happen not to occur in the program's output but would be constructed by reference to CANLIST if they did.

The pronoun specialist, then, has PREVREF and CANLIST which together constitute a pool of potential antecedents for pronouns. The specialist thus takes a cautious view

of anaphora, and does not authorise a pronoun whose interpretation requires a sophisticated comparison of the prominence of alternative antecedents or an evaluation of constraints entailed by the semantic context and, in particular, by the verb. The natural way to incorporate such calculations into any development of the present system would be as part of a procedure designed to interpret input: then a single procedure would be used both to understand anaphora in input, and to monitor its use in output.

However, at present the pronoun specialist authorises a proref pronoun if its referent coincides with the referent of one of the pool of antecedents. The antecedent of a relative must be in CANLIST, but the decision to use a relative pronoun is not one for the pronoun specialist; the feature {Relative} of a Ng is an "anticipated" feature and the pronoun specialist uses that as its justification for authorising a relative pronoun. The antecedent of other proref pronouns always occurs in PREVREF for the present range of discourse. The only deictic pronoun available to the system is "that". This pronoun directs the anaphora to the thing most recently mentioned (Chapter 3 section 3.2.1). The present version of the system notes when noun-groups preceding the antecedent might be confused with it, and used a deictic in such cases to select the nearer noun-group:

"You began the game by taking a corner, I took an adjacent one, and you took the one adjacent to that".

An earlier version used the deictic primarily when the {Proref...} Ng was embedded in a complex Ng and the antecedent lay before. The objective was to get a stressed or 'disjunctive' form of the pronoun, which was capable of standing independently within a complex Ng. We notice that:

"The game began with your taking the middle of an edge, I took an adjacent corner, and you took the middle of the edge adjacent to it and to the edge which you had just taken the middle of."

is greatly improved by substituting 'that' for 'it':

"The game began with your taking the middle of an edge, I took an adjacent corner, and you took the middle of the edge adjacent to that and to the edge which you had just taken the middle of.".

But this substitution is desirable only when the antecedent is outside the complex Ng and the anaphora has to be shot clear of the superordinate distractors: otherwise 'it' is preferable, as it is in:

"(He was wearing) a tie with several ketchup stains on it".

Prostring pronouns too must be authorised by the

pronoun specialist. The procedure makes its decision after examining the potential antecedents in PREVREF and CANLIST. It is cautious in licensing prostring pronouns, because the complex referring expressions constructed by the system become hard to understand if prostring dummies proliferate. For this reason the procedure does not incorporate Winograd's observation (1972: 161) that the antecedennt of 'one' is often found to have a contrastive modifier or qualifier:

"the large block...the small one"

"the large red block...the small (sc. 'red') one".

The existing rules already issue enough prostring licences, but the point of the 'contrast' rule could be only to increase the number of licences by admitting as antecedents contrasted noun-groups which are currently excluded as being too obscure.

The procedure uses CANLIST to find an antecedent in the same noun-group as the prostring but at a higher level of constituent structure:

"the middle of the edge opposite yours".

Here "yours", equivalent to "your one", picks up "edge". However the licence is refused if a distractor whose referent is of the same type (see p.248) intervenes. So in:

"You threatened me by taking the corner common to the edge adjacent to the corner which you had just taken and to the edge opposite..."

the procedure does not allow the second occurrence of "corner" to be replaced by "one", because "edge" comes between the antecedent "corner" and the current one, and "edge" and "corner" are both of the type LOCATION. After searching CANLIST, and provided no antecedent or inhibiting distractor has been found, the procedure searches PREVREF for an antecedent. Because the entries in PREVREF are constantly reviewed by the pronoun specialist, it now assumes that potential distractors have been accounted for: so if an entry in PREVREF offers an antecedent, a prostring is licensed without more ado.

A weakness in the procedure must now be explained. We have seen that CANLIST contains only referents, since no details are yet available of the words used in the noun - groups still under construction. So the procedure has to guess what word will be used to denote the antecedent's referent and to assume that this word will be available to be picked up by the prostring. But the antecedent may in fact be a pronoun. The antecedent of 'yours' in:

"(you took a corner, and I took) the one opposite yours"

is "the one opposite yours", which contains no word "corner". The example is, of course, perfectly acceptable

English, but it is probably wrong to regard "the one opposite yours" as the antecedent. The preceding noun-group "a corner" is better regarded as the antecedent of both prostrings in the example.

The procedure makes the same assumption about certain antecedents found in PREVREF, and so is open to the same criticism. In constructing:

"(You started the game by taking a corner, I took the opposite one, and you took) another one"

the antecedent of the second "one" is "the opposite one" not, as it probably should be, "a corner". The procedure's assumptions about the antecedent make it necessary to extend our notion of a prostring pronoun so that it can pick up not only explicit word-strings but also word-strings implicit in the meaning of the antecedent. Such an extension should not be made until it becomes necessary. It is made in the present procedure only to avoid the necessity to maintain more elaborate records in PREVREF and CANLIST, and, where the antecedent is still under construction, to obviate any necessity to reconsider the licence issued for a prostring in the light of alterations in the antecedent.

6.7.7 The restriction-finding specialist

6.7.7.1 Introduction

The restriction-finding specialist takes control of noun - group construction if the noun-group constructor has

determined that a simple proref pronoun, propernoun, common noun, or prostring pronoun does not constitute an adequate referring expression. This specialist is responsible for determining noun-group features in systems 71 - 82, though it will be apparent that the program never produces {Ordered}, system 74, or {Determined Deictic}, systems 71 and 72, noun-groups. The meaning of the features in systems 71 - 82 is quite clear, and so no attempt will be made to explain how the restriction-finding specialist determines the systemic choices in the particular context of noughts-&-crosses. We are concerned not with the semantics of this rather limited game, but with the principles which we can make precise and illustrate from the semantics. Therefore in the remainder of section 7.7 we shall examine not every variation illustrated in the examples, but only certain aspects of the procedure's work which have some more general interest.

The restriction-finding specialist attempts to add information to the noun-group in order to construct the most economical adequate referring-expression possible. When it finds that one of its repertoire of alternatives fills the bill, it adds the corresponding features to the noun-group feature list. So if it decides that the best way to refer to a corner is with the aid of a relative clause:

"...which you had just taken"

it marks the noun -group {Qualified Clausequal}. In this

way the noun-group constructor ultimately has available the complete feature-set.

It is not easy to ensure that the procedure always comes up with the simplest solution. It runs through its options in an order which generally produces reasonable results, but each option is investigated by its own specialist procedure which may in many cases wish to call recursively the noun-group constructor and restriction - finding specialist. Some rather simple checks are made to ensure that this recursion does not get out of hand. For example, one of the simplest ways of identifying a square is to say what it is opposite; however, the attempt to use this simple method may have to be abandoned if it proves impossibly complex to identify the square we wish to use as a reference point. In general the procedure has resources adequate to ensure that it is never floored, but its complexity often makes it hard to foresee what restriction it will concoct for a particular referring expression. Such apparent unpredictability may be a feature of any productive program whose capacity is more than trivial.

6.7.7.2 The anaphoric adjective "other".

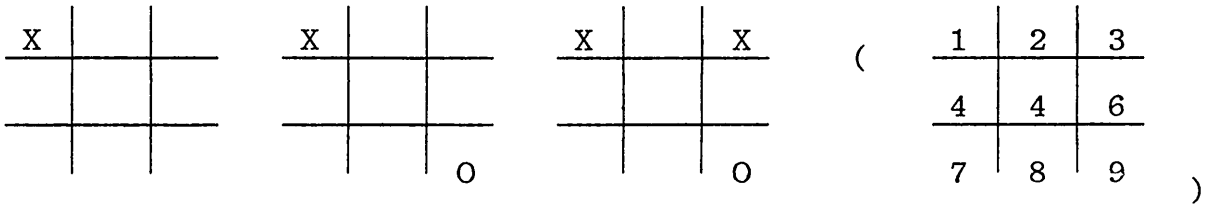
One of the adjectives the restriction finding specialist may recommend is the anaphoric adjective 'other', whose idiosyncrasies are considered in this section.

It happens sometimes that the equivalence class XEQCL of

the current referent is a subset of some larger set, all the other members of which have been mentioned recently. In this situation 'other' is a modifier adequate to restrict to the XEQCL subset the scope of the noun used to denote the subset. Of course, the noun itself may be replaced by a prostring pronoun or be left out altogether if the context permits. For example:

> "The game began with my taking a corner, you took the opposite one, and I threatened you by taking another."

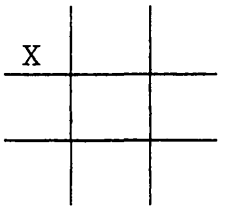
describes a game which went as follows:

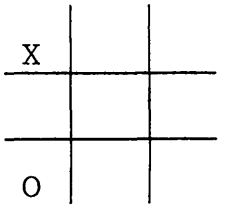


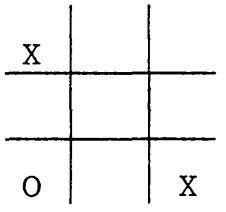
The sketch on the right is a reminder of the names of the squares. In the example "another" is equivalent to "another corner". We can see that its referent is square 3 and its XEQCL was {square 3, square 7}. The XEQCL was thus a subset of the larger set of corners, all the other members of which had just been mentioned.

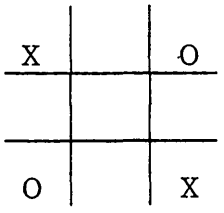
The task of the restriction-finding specialist is to decide when the context permits the use of 'other'. It takes the set of entities denoted by the head-noun of the current referring expression, or, if the noun has been

replaced by a prostring or omitted, by the antecedent head - noun, and it checks that every entity in the complement of XEQCL in this set has been mentioned recently. 'Recently' is rather narrowly interpreted to mean either within the current sentence or at least within the scope of the PREVREF stack. In the example, squares 1 and 9 had been mentioned within the sentence. This criterion excludes the use of 'other' in some circumstances where it might be acceptable. For example, it would not be used in description of a game which went:







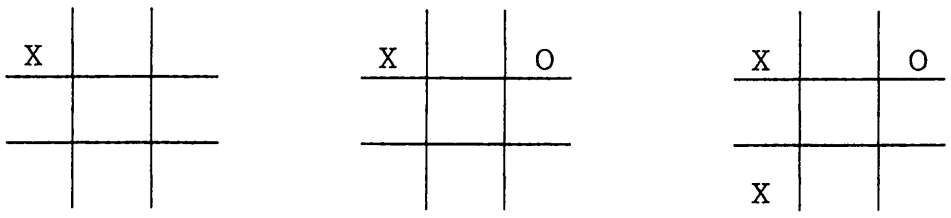


> "You began the game by taking a corner, and I took an adjacent one. You threatened me by taking the one opposite the one you took first, and I took the other.".

This use of 'other' is perhaps acceptable, but the procedure avoids it in favour of other more explicit alternatives on the grounds that the allusions to the rest of the 'corners', namely squares 1, 7, and 9, do not meet

the criterion of recency.

The procedure in fact goes rather further than has been explained so far. It does not simply test whether the appropriate conditions have been met, but sometimes manipulates the context so that they will be. Consider a game which went:



After move 2, the XEQCL of square 7 is just {square 7}. The set of corners is {square 1, square 3, square 7, square 9} of which only squares 1 and 3 have been mentioned. The unmentioned residue is thus {square 7, square 9}, with which XEQCL does not coincide. Therefore the rules so far stated would not sanction the use of 'other' in the referring expression for square 7, and the description might be:

> "You began the game by taking a corner, I took an adjacent one, and you threatened me by taking the one opposite that.".

In fact, however, the program produces:

> "You began the game by taking a corner, I took one of the adjacent ones, and you took the other.".

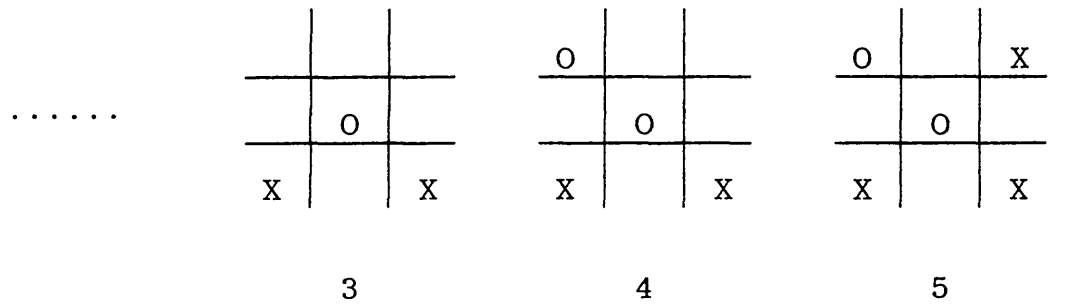
In this sentence 'the other' is equivalent not to 'the

other corner' which, as we have seen, the rules would properly ban, but to 'the other (one) of the adjacent corners'. The construction 'one of...other' must be taken as a whole: when 'other' closely follows such a partitive expression, the set whose unmentioned residue is now meant is the same set as was referred to in the partitive expression. That is, if we regard 'the other' as equivalent to 'the other of the adjacent corners', the set 'the adjacent corners' is the set denoted by the first occurrence of that expression, in 'one of the adjacent corners', namely {square 3, square 7}. Square 3 has been mentioned, so the XEQCL of square 7, namely {square 7}, coincides with the unmentioned residue, and the rules sanction 'other'.

It is worth emphasising the fact that in order to be able to use 'other' in allusion to square 7, a partitive construction must be used in the allusion to square 3. This identifies the set whose residue is later picked out by 'other'. The anaphora to this set by-passes any updating of the hearer's model; after the description of move 2 squares 3 and 7 do not form an equivalence class. In any case they could not be referred to in the description of square 7 as 'the adjacent ones' because the hearer would be uncertain what they were 'adjacent' to. The point to which they were adjacent would have to be made explicit, as it was in the first alternative description above. So 'the other' is anaphoric, and its construction makes little

use of the apparatus of conventional restriction-finding.

We have seen that the restriction-finding specialist must recommend a partitive construction, 'one of the adjacent corners', where a simple indefinite would have been enough, 'an adjacent corner', in order to mark out the antecedent for 'other' which will follow in a later noun-group. The procedure is again cautious in arranging 'one of...other' constructions, and requires that both referring expressions occur within the current sentence. Otherwise the partitive form is not used, and 'other' then cannot be used in this way. The procedure finds out what referring expressions will occur by examining the semantic representation of the sentence made by the sentence designer. Obviuosly this is not really a suitable check except for rather simple cases, because it is not generally possible to tell what referring expression will be used to denote a particular referent in the sentence design. Suppose, for example, that the current sentence was to describe the fourth and fifth moves shown here:



The semantic representation of the sentence would include

the information that squares 1 and 3 were to be mentioned, but not that the situation permitted us to use:

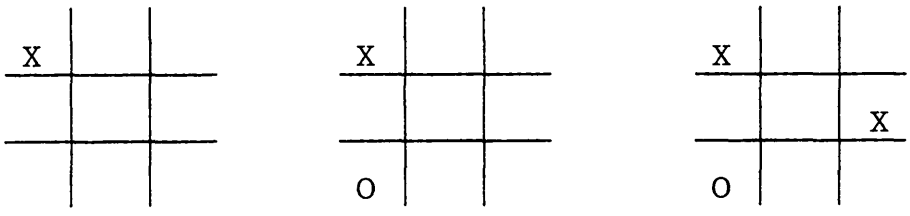
> "the end of one of the diagonals...the end of the other"

to refer to them. The procedure includes a simple ad hoc check for such cases. In general, though, the use of the construction 'one of...other' implies a somewhat interesting departure from the strict left-to-right order of constructing utterances, and requires some capacity to construct two expressions in parallel, or to revise constructions already completed.

6.7.7.3 Implicit antecedents

Sometimes the context suggests an idea so strongly, but without mentioning it explicitly, that the speaker can subsequently refer to this implied antecedent and be understood. The present program makes use of implied antecedents in only one situation, but displays the phenomenon in two slightly different ways.

The description of the following three moves:



in one version of the program is:

"The game began with your taking a corner, I took an adjacent one, and you took the middle of the opposite edge".

We notice that what the 'opposite edge' is opposite is neither of the two squares mentioned so far, but rather is the edge on which both these squares lie. This edge is sufficiently prominent in the context to be able to act as a reference point.

6.7.7.4 The anaphoric adjective 'same'

The second way the program uses implied antecedents involves the adjective 'same'. This adjective is anaphoric. It restricts the scope of its referring expression the referent of its antecedent, and so may appear as a somewhat emphatic alternative to a proref pronoun:

"John saw a flying saucer. Mary the same one (it) too".

However, the referring expression with 'same' may have a head-noun, unlike the proref pronoun. This may mean that in cases where anaphora is impossible for a simple pronoun, 'same' may be usable because its head-noun is a content word and so helps the hearer to understand what antecedent is intended. So the program produces:

"I began the game by taking a corner, you took an adjacent one, and I took the middle of the same edge".

As before, the 'edge' involved is the edge upon which the

last-mentioned squares lie, but we see that the program could not say:

*"...and I took the middle of it"

because the antecedent is insufficiently apparent. Thus some marginal or unacceptable uses of pronominal anaphora may be substituted for by a full referring expression with the anaphoric adjective 'same'.

The program uses 'same' only in such anaphora to implicit antecedents. It never uses a full referring expression with 'same' in place of a proref pronoun if the antecedent is explicit, because the motivation for such substitution is unclear in this universe of discourse. The restriction finding specialist regards 'edge' as implicit in the context if the last two squares mentioned were on the edge in question. If 'same' is to be used, as in '...the same edge', the procedure applies much the same criteria as the pronoun specialist applied in the licencing of anaphora. Both of the squares which identify the implicit edge must have been mentioned recently enough to be in PREVREF. Furthermore, the procedure does not use the anaphoric adjective if CANLIST contains a confusable distractor. This is exactly the same check as the pronoun specialist makes before recommending a prostring pronoun. Here it prohibits a construction such as the following:

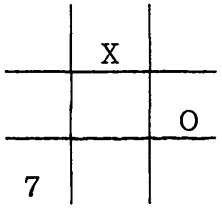
* "...the middle of an edge adjacent to the same edge"

because the implicit antecedent of 'same' is obscured by

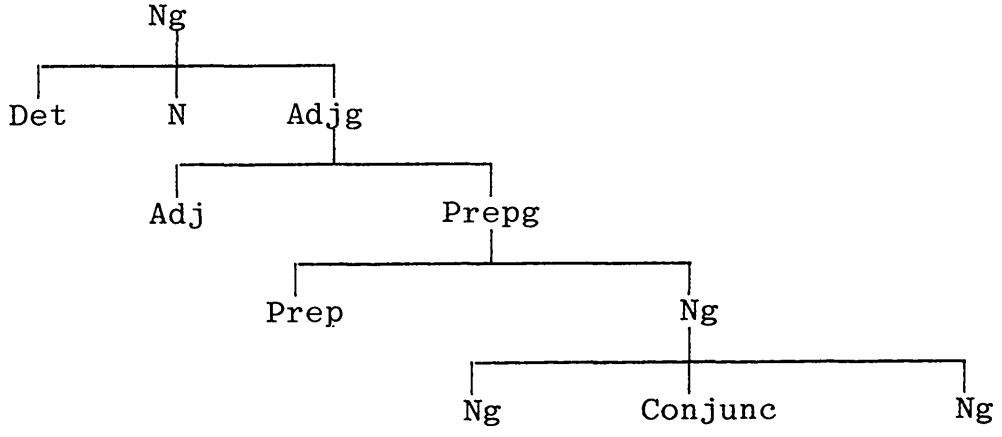
the intervention of 'an edge adjacent...'.

6.8 Making Preposition-groups

The preposition-group is a simple structure, comprising just a preposition and a noun-group. In the program's range of output it is simpler still, since the preposition is always a single word rather than a complex expression such as 'instead of', 'up to', or 'in back of'. Furthermore, it never produces coordinate preposition groups. To refer to square 7 in the following situation:



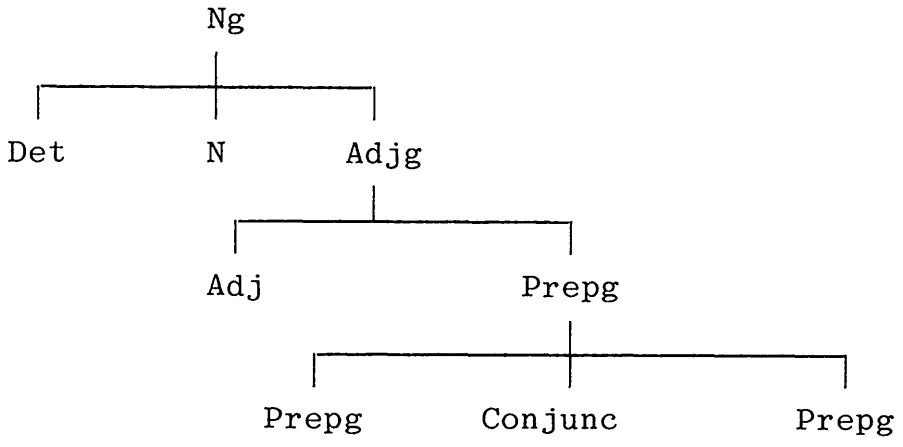
it produces:



"the corner common to the edge opposite the square

X had just taken and the one opposite the square O had just taken"

rather than:



"the corner common to...and to...".

It should be noted that the preposition-group constructor could be extended to cope with coordinate preposition - groups, but only at the cost of an amendment to the present design. As we have seen, the constructor does not select the required preposition for itself; the preposition is selected by the restriction-finding specialist and passed to the preposition-group constructor by way of the feature list. But a coordinate preposition-group may have different prepositions with each coordinated constituent, as:

"(a holiday) in the Highlands and on Skye".

This cannot be expressed by a feature of the {Complex} preposition-group, at least in any form as simple as that used at the moment. It seems likely that the preposition-group constructor should select the correct preposition

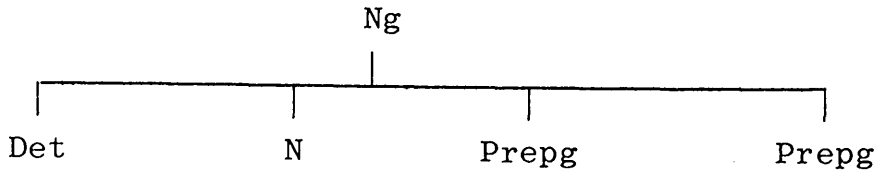
for itself, taking account of the relation to be expressed and the character of the prepositional object. The extent and form of the guidance which the constructor should be given, whether by a feature or otherwise, is debatable.

It is perhaps worth reiterating here that the preposition-group constructor has no special duties in connection with 'dangling' prepositions, as:

"...(the middle) of".

The feature {Danglingp} is supplied to the preposition-group constructor by the procedure which calls it, and the realisation rules ensure that in this context no prepositional object is realised. No action is required from the constructor.

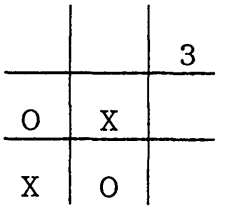
The only decision made by the preposition-group constructor is the stylistic one which was mentioned in the first chapter. We want to avoid following one completed preposition-group with another because in the present universe of discourse it makes for obscurity. The procedure therefore carries out a simple check whether the constituent immediately to the left in surface structure is a preposition-group. The check detects such a constituent at any level at or below the level of the current constituent. That is, the check is designed to exclude not only:



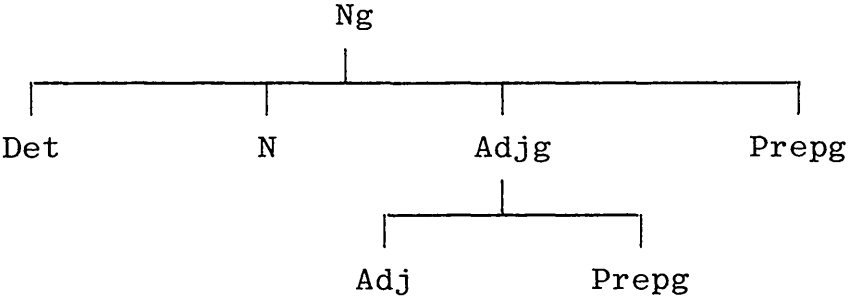
as:

* "the end of X's diagonal between the empty edges"

to refer to square 3 in:



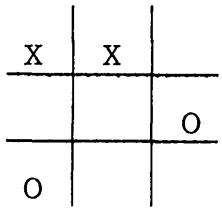
but also an expression where the preceding preposition - group is at a lower level of constituent structure. It thus excludes:



as:

* "the edge adjacent to X's of which O had just taken the middle"

to denote the right-hand edge, squares 3, 6, 9, in:



This check was incorporated into the program at an early stage of development to exclude the kind of example just illustrated. However, it is clear that a purely structural criterion is inadequate. If the meaning of the adjacent constituents makes it obvious what head-noun the second qualifier applies to we may have perfectly acceptable examples of double post-nominal qualifiers, as:

"the girl with the stammer in the corner".

We know that it is 'the girl', not 'the stammer', which is 'in the corner' because 'the stammer in the corner' is nonsense. The check, then, must have a semantic component to examine the potential for confusion detected by the structural check.

The present version of the program incorporates a re-organised restriction-finding specialist which happens never to recommend a construction which would be rejected by the structural test in the preposition-group constructor. However, the test is retained because in any development of the system such a test, with an added semantic component, would probably be required at this point in the productive process.

6.9 Making Adjective-groups

We saw earlier in this chapter that the restriction-finding specialist has all the responsibility for determining the characteristics of the adjective-groups. By the time the adjective-group constructor is invoked, all the constituents of the group are in existence, ready to be assembled to form an adjective-group. The constructor is therefore very simple; it is of little intrinsic interest, and in the interests of computational efficiency the realisation and structure building rules given in the last chapter have been by-passed with a programming short cut. An extension of the rules to cope with a fuller range of adjective - groups, as discussed in chapter 4, would make it desirable to reinstate the full grammatical apparatus in order to test it properly. For the moment, however, the procedure simply places the adjective in position and, if the group is {Complex}, the focussing phrase after it. If the group is {Coord}, the constructor invokes itself recursively, to construct each coordinated constituent in the expected way.

6.10 Making Words

The program has only a rather simple word-maker, which incorporates just those rules of morphology and idiom which the program's output requires.

As we saw in chapter 4 section 6, the word systems for prepositions, adverbs, and binders include the required

words as features. In these cases, then, the word-maker simply picks out the feature in question from the feature-set of the constituent. In the case of the negative adverb 'not', it additionally checks to see if the preceding word is a verb with the feature Grammatical ; if it is, the procedure idiomatically attaches the adverb to the verb, giving such forms as "hasnt" in:

"The game hasnt finished".

If the word is a pronoun, the procedure invokes a specialist pronoun finder. This specialist uses the features of the pronoun to select the particular deictic, relative, possessive, or unmarked form required, and, where appropriate, to pick the form necessary for the surface case of the pronoun, 'she' or 'her' for example. The prostring pronoun 'one' requires special attention since in the plural it takes '-s' as though it were a conventional noun. The possessive determiners 'my, your, his,...' are selected in much the same way as pronouns, the form being determined by the number, person, and, in the third person singular, the gender features of the word.

The articles 'a' and 'the' are distinguished by reference to the Definite feature, and the idiomatic addition of '-n' to 'a' before a vowel, as in 'an egg', is specially catered for. The conflation of 'an' with 'other' to yield 'another' is likewise the subject of a special check. The genitive morpheme '-s' is added, as

"Claribel's", but no account is taken of its behaviour when appended to a noun already ending, for example, in "-s" as "Nils' Danish accent" as such nouns have not occurred in the program's output. The word systems specially distinguish number-words, 'one, two,...', as in 'one of your lines'; the word makes invokes a specialist for these words to count the objects denoted and then to fetch the required number word from a list.

The making of nouns and verbs has been left to last as these items are dealt with somewhat differently from the rest. Earlier in this chapter we saw that every entity which the system could refer to belonged to a class of entity in a hierarchy of object names (page 248). Each class marked {---} in the hierarchy has a name attached to its representation in the computer, and the name is the word selected as the head-noun by the noun-group maker. If the word is a propernoun the word maker need take no action beyond fetching the word, since the word is in its final form already, as "Claribel". Common nouns, however, are stored in their plural form, 'edges, squares,...' in order to avoid a technical programming problem. The word maker therefore removes from these words the plural morpheme if the word is {Sg}. In so doing, it takes no account of the complex rules of plural formation in English, because the words it uses all form the plural with simple suffixed '-s'.

The verb words output by the program are constructed by a set of specialists. Some of the features attached to the verb-group constituent are, for programming convenience, the names of specialist procedures. The verb -group specialist therefore first orders the unordered feature-set, and then traverses the ordered feature-set, invoking each procedure as it is encountered. Each procedure makes the appropriate addition or alteration to the site of the verb word, relying on its predecessors, if any, to have prepared the way: a somewhat similar technique was used by Longuet-Higgins in a pilot program (1970, unpublished). An example will demonstrate what happens. Suppose the required word is 'has' as in:

"The game has not finished".

The constituent has the unordered feature-set

{3 Sng Tensed Present Vg Have Aspectual Grammatical}

which the verb-group maker orders

{Have Aspectual Grammatical Present Tensed Vg 3 Sng}

Of these features, Grammatical and Present are procedure names. The first relies upon the ordering operation to have thrown the required grammatical verb to the front of the feature list, and so in this case picks up 'have'. The second procedure, Present, consults the lexicon to see whether 'have' is a weak verb whose formation follows simple morphological rules, or a strong verb whose forms are irregular and are given in the lexicon. 'Have' is a

strong verb, and so the procedure picks the

{3 Sng Present}

form 'has'. If the verb had been weak, the procedure would have invoked a specialist suffix procedure which applies the right rules of morphology insofar as the program's output requires them. A fuller procedure is described by Winograd (1972: 74).

Weak verbs have simple past, and past participle, forms in '-ed'. The procedure simply adds ' -ed ', as 'blocked', unless the root itself ends in ' -e ', when we require the elimination of it, as 'complete, completed'. The procedure likewise eliminates the final ' -e ' of the root when suffixing ' -ing ', as in 'complete, completing', except when the root is just one syllable long, as 'be, being'. The rules for doubling a consonant before a suffix which starts with a vowel have been greatly over-simplified: the procedure looks specifically for ' -m ' or ' -n ' as the root termination, and doubles it as needed, thus giving 'win, winning' but 'wash, washing'. This rule is plainly inadequate, since polysyllabic words should be distinguished, 'threatening' not 'threatenning'.

It will be apparent that the word-making procedures and the suffix specialist in particular are somewhat ad hoc, and contain nothing new. The morphology of words is a subject of considerable interest in its own right,

particularly when considered in relation to a system which might produce or recognise the spoken word. It is, however, a subject rather beyond the intentions of the present enterprise.

Note 1 (to follow chapter 6)

Calculation of the equivalence class of a part of the board.

The objective is to find the set of board parts which are tactically equivalent to the given board part in the momentary board situation. The method adopted is to apply in turn each of the operations of the symmetry group of a square matrix, except of course Identity, to the board. Any operation which is found to take each square occupied by a particular player into some square occupied by the same player, and to take each empty square into some empty square, evidently takes the board situation given into an equivalent one. So the board-part into which the given board-part is taken by this operation is added to the equivalence class of the given board-part.

APPENDIX 1

```
: .PLAY;
 I AM GOING TO START THE GAME
    .    .
    .    . P
............
    .    .
    .    .
............
    .    .
    .    .


YOUR MOVE: 1
    .    .
 A  .    . P
............
    .    .
    .    . P
............
    .    .
    .    .


YOUR MOVE: 9
    .    .
  A .    . P
............
    .    .
    . P  . P
............
    .    .
    .    . A

YOUR MOVE: 4

 I HAVE WON THE GAME

    .    .
 A  .    . P
............
    .    .
 A  . P  . P
............
    .    .
 P  .    . A



: .SPOUT;
 THE GAME STARTED WITH MY TAKING A CORNER, AND YOU TOOK AN ADJACENT
ONE. I THREATENED YOU BY TAKING THE MIDDLE OF THE EDGE OPPOSITE
THAT AND ADJACENT TO THE ONE WHICH I HAD JUST TAKEN BUT YOU BLOCKED
IT AND THREATENED ME. I BLOCKED YOUR DIAGONAL AND FORKED YOU. IF
YOU HAD BLOCKED MINE, YOU WOULD HAVE FORKED ME, BUT YOU TOOK THE
MIDDLE OF THE EDGE OPPOSITE THE CORNER WHICH I TOOK FIRST AND THE
ONE WHICH YOU HAD JUST TAKEN AND SO I WON BY COMPLETING MY DIAGONAL.
```



```
:
```

EXAMPLE 11 21.01 6 APRIL 1973

ACD	PROTEUS
7	4
9	8
2	5
1	6

THE GAME BEGAN WITH YOUR TAKING A CORNER, AND I TOOK THE MIDDLE OF AN ADJACENT EDGE. YOU THREATENED ME BY TAKING THE END OF THE ONE ADJACENT TO THE CORNER WHICH YOU HAD JUST TAKEN AND TO THE ONE WHICH I HAD JUST TAKEN THE MIDDLE OF BUT I BLOCKED THAT, AND YOU TOOK THE SQUARE OPPOSITE THE ONE WHICH I HAD JUST TAKEN. I TOOK THE MIDDLE OF THE BOARD AND THREATENED YOU. IF YOU HAD BLOCKED MY LINE, YOU WOULD HAVE THREATENED ME, BUT YOU TOOK THE CORNER ADJACENT TO THE ONE WHICH YOU TOOK FIRST AND SO I WON BY COMPLETING MY LINE.

[SPOUTPUT]
EXAMPLE 2 13.29 29 SEP 1973

```
  PROTEUS      ACD
     1          7
     9
```

I STARTED THE GAME BY TAKING A CORNER, YOU TOOK AN ADJACENT ONE, AND I THREATENED YOU BY TAKING THE ONE ADJACENT TO THAT. THE GAME HASNT FINISHED.

```
  PROTEUS      ACD
     1          9
     7
```

THE GAME STARTED WITH MY TAKING A CORNER, YOU TOOK THE OPPOSITE ONE, AND I THREATENED YOU BY TAKING ANOTHER. THE GAME HASNT ENDED.

```
  ACD      PROTEUS
   1          3
   7
```

THE GAME BEGAN WITH YOUR TAKING A CORNER, I TOOK ONE OF THE ADJACENT ONES, AND YOU THREATENED ME BY TAKING THE OTHER. THE GAME HASNT ENDED.

[SPOUTPUT]
EXAMPLE 3 15.36 29 SEP 1973

```
   PROTEUS      ACD
      1          3
      2
```

THE GAME BEGAN WITH MY TAKING A CORNER, YOU TOOK AN ADJACENT ONE, AND I TOOK THE MIDDLE OF THE SAME EDGE. THE GAME HASNT ENDED.

```
   ACD      PROTEUS
    1         2
  ⌒ 3
```

THE GAME BEGAN WITH YOUR TAKING A CORNER, AND I TOOK THE MIDDLE OF AN ADJACENT EDGE. IF YOU HAD TAKEN THE CORNER OPPOSITE THE ONE WHICH YOU HAD JUST TAKEN YOU WOULD HAVE THREATENED ME BUT YOU TOOK THE ONE ADJACENT TO THE SQUARE WHICH I HAD JUST TAKEN. THE GAME HASNT FINISHED.

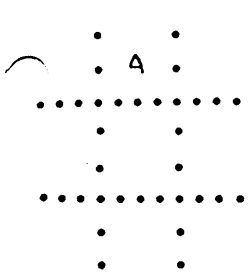
```
   PROTEUS      ACD
      1          7
      6
  ⌒
```

THE GAME BEGAN WITH MY TAKING A CORNER, YOU TOOK AN ADJACENT ONE, AND I TOOK THE MIDDLE OF THE VACANT EDGE. THE GAME HASNT FINISHED.

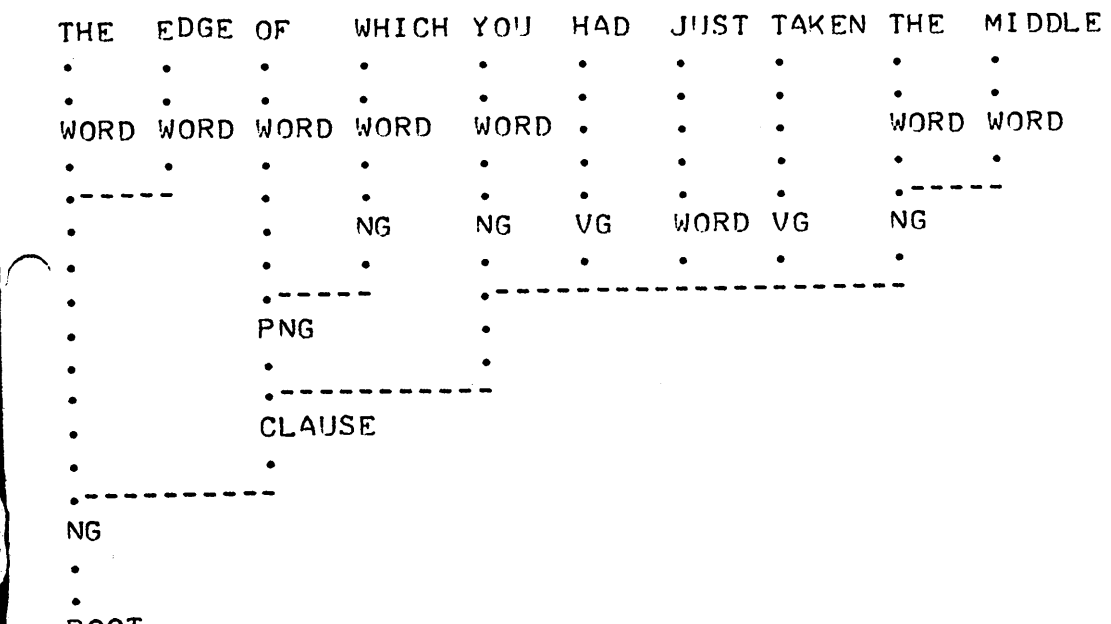
2300

25 SEP 1973

```
: COMMENT 'REFER TO THE TOP EDGE.';
: 0->LINEPOS; 0.BPRINT;
```




```
: ROOT.TREEPR;
```




```
:
```

APPENDIX 2

(This appendix has been placed at page 369)

APPENDIX 3

Appendix 3

Directory of function location within files

Function Name	File Name
ACTOR	CLAUSE FUNCTION FNS
ADDTO	TREEFNS
ADJACENT	LOCFNS
ALL	HCLHFNS
ANIMATEQ	NFEATFNS
APPENDIX	CLAUSE FUNCTION FNS
APPTESTLIST	ODDFNS
ASSESS	DESIGN0
BELOW	TREEFNS
BFETCH	BFETCH
BFIND	BFETCH
BLOCK	GAMEFNS
BPRINT	SPIELEN
BRACKETREAD	MACROS
BUNDLE	SB FNS
CHECKDE	CHECKMEN
CHECKMEN	CHECKMEN
CHOICEOF	ODDFNS
CHOPOFF	SUFFIX
COINCIDE	SETFNS
COMPLEMENT	SETFNS
CONDENSE	SETFNS

Appendix 3

Function Directory (continued)

Function Name	File Name
CONSNODE	TREEFNS
CONSONT	V SETUP
COORDS	BOARD
COORDS2N	BOARD
DEAD	HCLHFNS
DEFQ	DETERMIN
DEPTH	TREEFNS
DESIGN1	DESIGN1
DETERMIN	DETERMIN
DISPLAY	ODDFNS 2
DRAW	GAMEFNS
EMPTY	HCLHFNS
END1	LOCFNS
ENVIR	CLAUSE FUNCTION FNS
EQCL	EQCL
EVARS (macro)	ENTITY
EVERY	HCLHFNS
FETCH	V SETUP
FORK	SPIELEN
FREE	HCLHFNS
GENITIVE	NGFNS
GET	SETFNS
GETCOND	FTR FNS
GIVEN	ODDFNS

Appendix 3

Function Directory (continued)

Function Name	File Name
GOAL	CLAUSE FUNCTION FNS
GRAMMATICAL	VG FNS
HOWMANY	HCLHFNS
INITR	CLAUSE FUNCTION FNS
INTERSECT	SETFNS
ISA	HCLHFNS
ISIN (macro)	MACROS
LASTWO	ODDFNS 2
LISTDELETE	SETFNS
LMAP	EQCL
LOCDESCR	LOCDESCR
LOCD1 - LOCD5	LOCDESCR FNS
LOCD6 - LOCD9	LOCDESCR FNS 2
LOCD10	LOCD10
LOGGEDON	ODDFNS
LRRP	LRRP
MAKEADJG	MAKEADJG
MAKENG	MAKENG
MAKEPNG	MAKEPNG
MAKERELN	MAKERELN
MAKES	MAKES
MAKESFN1	MAKESFN1
MAKESIF	MAKESIF
MAKESTR	ODDFNS

Appendix 3

Function Directory (continued)

Function Name	File Name
MAKEVG	MAKEVG
MAKEWORD	MAKEWORD
MATREQ	BOARD
MEMBER	SETFNS
MIDDLE	LOCFNS
MOVENO	ODDFNS 2
NCREV	SETFNS
NIMMT	ODDFNS 2
NO	HCLHFNS
NODEPR	TREE DISPLAY FNS
NONMAP	EQCL
NUMBEROF	HCLHFNS
NUMBERWORD	NGFNS
OCCUPIED	HCLHFNS
OF (macro)	MACROS
OPEN	HCLHFNS
OPPONENT	ODDFNS 2
OPPOSITE	LOCFNS
PASTPART	VG FNS
PERFORM	FTR FNS
PERSONQ	NFEATFNS
PLAN	SPIELEN
PLAY	SPIELEN
PRESNT	VG FNS

Appendix 3

Function Directory (continued)

Function Name	File Name
PRESPART	VG FNS
PROCESS	CLAUSE FUNCTION FNS
PRONOMINALISE	NGFNS
PRONOUN	NGFNS
PROPERNOUN	NFEATFNS
PROPERTIES	V SETUP
PROSTRING	NGFNS
PROVAL	CONJUNC TABLE
PTDF	TREEFNS
PTDL	TREEFNS
PTKINDLE	TREEFNS
PTLMOST	TREEFNS
PTLT	TREEFNS
PTPV	TREEFNS
PTRT	TREEFNS
PTS	TREEFNS
PTU	TREEFNS
PTYBRO	TREEFNS
READMOVES	RDMOVES
READTO	ODDFNS
REALISE	FTR FNS
RELATED	LOCFNS
REMOTE	VG FNS
RESET	BOARD

Appendix 3

Function Directory (continued)

Function name	File name
RETROVAL	CONJUNCT TABLE
ROLELIST	ODDFNS 2
S (macro)	MACROS
SB1	SB FNS
SB2	SB FNS
SENTPR	TREE DISPLAY FNS
SG	NFEATFNS
SMAP	EQCL
SOME	HCLHFNS
SOMEHOW	SETUP
SPATIALRELATION	SPATREL
SPATPART	SPATREL
SPIT	V SETUP
SPOUT	SPOUT
SQVAL	SPIELEN
SUFFIX	SUFFIX
TAKEFN	ODDFNS 2
TENSE	VG FNS
TESTCOND	FTR FNS
THREATEN	SPIELEN
TO	VG FNS
TQ	TREEFNS
TREEPR	TREE DISPLAY FNS
UPDATE1	UPDATE1
USEFUL	HCLHFNS

Appendix 3

Function Directory (continued)

Function Name	File Name
VACANT	HCLHFNS
VERBTYPE	V SETUP
VOWEL	V SETUP
WHICH	HCLHFNS
WIN	GAMEFNS
==> (macro)	MACROS
--> (macro)	MACROS
£ (macro)	MACROS
:+ (macro)	MACROS
↑↑ (macro)	MACROS
↑↑↑ (macro)	MACROS

```
[BFETCH]                          [ 20.46 11 MAY 1974]
DTRACK 217     CREATED    20.44    11  5 1974


 1   FUNCTION BFIND LIST ITEM => SUCCESS;
 2   0->SUCCESS;
 3   L1: IF LIST.GIVEN THEN
 4     IF (LIST.HD,EQUAL(ITEM)) THEN;
 5     ELSEIF LIST.HD.ISLIST AND (ITEM,ISIN(LIST.HD)) THEN;
 6     ELSE LIST.TL->LIST; GOTO L1;
 7     CLOSE;
 8     1->SUCCESS; LIST; RETURN;
 9   CLOSE;
10   END;
11
12   FUNCTION BFETCH LIST ITEM => SUCCESS;
13   VARS PT PT1 ST;
14   IF LIST.NULL THEN 0->SUCCESS; EXIT;
15   LIST->PT;
16   L1:
17     IF (PT.HD,EQUAL(ITEM)) THEN;
18     ELSEIF (PT.HD.ISLIST) AND (ITEM,ISIN(PT.HD)) THEN;
19     ELSEIF PT.TL.GIVEN THEN PT->PT1; PT.TL->PT; GOTO L1;
20     ELSE 0->SUCCESS; RETURN;
21     CLOSE;
22   COMMENT  ' PT.HD IS WHAT WE WANT.';
23   PT.HD->ST;
24   IF (ST,EQUAL(LIST.HD)) THEN
25    IF LIST.TL.GIVEN THEN
26       LIST.TL.HD->LIST.HD; LIST.TL.TL->LIST.TL;
27    ELSE 0->LIST.HD; IDENTFN->LIST.TL;
28     CLOSE;
29   ELSE PT.TL->PT1.TL;
30   CLOSE;
31   1->SUCCESS; ST;
32   END;
33
```

```
[BOARD]                          [ 20.32 11 MAY 1974]
DTRACK 217     CREATED    20.30    11  5 1974


 1   FUNCTION COORDS N;
 2   VARS I; (N-1)//3->I->N; (N-1),(I-1);
 3   END;
 4
 5   FUNCTION COORDS2N I J; (((J+1)*3)+(I+1))+1; END;
 6
 7   VARS BOARDV;
 8   NEWARRAY([%-1,1,-1,1%],
 9   LAMBDA I J; VARS XT;
10   COORDS2N(I,J)->I;
11   IF (I,ISIN([1 3 7 9])) THEN CORNERS
12   ELSE SQUARES; CLOSE;
13   ->XT;
14   CONSENTITY([%XT%],NIL,[%I%],[%I%]);
15   END)->BOARDV;
16   VARS DUMMY; CONSENTITY([%LOCATION%],"BOARD",NIL,NIL)->DUMMY;
17   EVARS BOARD DUMMY;
18   MACRO SQUARE;
19    ([BOARDV(COORDS(]<>[%.NUMBERREAD%]<>[))])==>; END;
20
21   FUNCTION RESET; VARS I; 1-> I;
22   FORALL I 1 1 9;
23   [%I%]->CONTENTS(I.COORDS.BOARDV); CLOSE;
24   LOOPIF .STACKLENGTH>0 THEN .ERASE; CLOSE;
25   END;
26
27   FUNCTION MATREQ FN;
28   VARS I J X Y;
29   -2 ->J;
30   LOOPIF (J+1->J; J=<1) THEN;
31    -2->I;
32    LOOPIF (I+1->I; I=<1) THEN;
33    TL(£BOARDV(I,J))->X;TL(£BOARDV(FN(I,J)))->Y;
34    IF X=Y THEN;
35     ELSEIF X.NULL THEN 0; RETURN;
36     ELSEIF Y.NULL THEN 0; RETURN;
37     ELSEIF NOT(X.HD=Y.HD) THEN 0; EXIT;
38      CLOSE;
39     CLOSE;
40   1;
41   END;
42
```

```
[CHECKMEN]                          [ 20.35 11 MAY 1974]
DTRACK 217    CREATED   20.32   11  5 1974


 1   FUNCTION CHECKDE REF XEQCL;
 2   IF (REF,ISIN(CHECKDE.FNPROPS.TL)) THEN 1; EXIT;
 3   VARS COMPLEQCL;
 4   IF COMPLEQCL.ISLIST THEN;
 5   ELSE COMPLEMENT(XEQCL,£HD(TYPE(HD(REF))))->COMPLEQCL;
 6   CLOSE;
 7
 8   COMMENT 'SEE IF EVERY MEMBER OF COMPLEQCL HAS BEEN MENTIONED,'
 9           ' 1 - IN ROOT.PPSELF.';
10   VARS ST SVCOMPL;
11   COMPLEQCL->SVCOMPL;
12   [%APPLIST(WHICH(ROOT.PPSELF,ISLIST),
13    LAMBDA X; X.HD,X.TL.HD; END)%]
14   ->ST;
15
16   L1:
17    LOOPIF ST.GIVEN THEN
18     IF (ST.HD,ISIN(COMPLEQCL)) THEN
19      LISTDELETE(COMPLEQCL,ST.HD)->COMPLEQCL;
20      IF COMPLEQCL=NIL THEN
21       IF ALL(REF,LAMBDA; ISIN(ST.TL); END) THEN 1; RETURN;
22       ELSEIF SVCOMPL.GIVEN THEN; GOTO L2;
23       ELSE 0; RETURN;
24       CLOSE;
25      CLOSE;
26     CLOSE;
27     ST.TL->ST;
28    CLOSE;
29
30   IF SVCOMPL=NIL THEN 0; EXIT;
31   L2:
32    SVCOMPL->COMPLEQCL; NIL->SVCOMPL;
33    COMMENT ' 2 - IN PREVREF.';
34    NIL->ST;
35    APPLIST(PREVREF,LAMBDA X;
36     IF X.GIVEN THEN X.HD->X;
37      LOOPIF X.GIVEN THEN X.DEST->X; -->ST; CLOSE;
38     CLOSE;
39    END);
40    GOTO L1;
41   END;
42
43
44   FUNCTION CHECKMEN REF XEQCL;
45   IF NUMBEROF(REF)=1 AND NUMBEROF(XEQCL)=2 THEN;
46   ELSE 0; EXIT;
47
48   LISTDELETE(XEQCL,REF.HD)-->CHECKDE.FNPROPS.TL;
49   COMMENT 'ARE BOTH MEMBERS OF XEQCL GIVEN IS ROOT.PPSELF 0.';
50   VARS ST; NIL->ST;
51   APPLIST(ROOT.PPSELF,LAMBDA X;
52    IF X.ISLIST THEN X.HD-->ST; X.TL.HD-->ST; CLOSE;
53    END);
54   IF ALL(XEQCL,LAMBDA X; X,ISIN(ST); END) THEN
55    "MEN"-->FLIST; 1;
56   ELSEIF (XEQCL.HD,ISA(LINES)) AND
```

```
57      ALL(XEQCL,LAMBDA X; £X.TL.HD,ISIN(ST); END) THEN
58      "MEN"-->FLIST; 1;
59    ELSE 0;
60    CLOSE;
61    END;
62
```

```
[CLAUSE FUNCTION FNS]                          [ 20.42 11 MAY 1974]
DTRACK 217    CREATED   20.39   11  5 1974

 1   FUNCTION ACTOR;
 2   IF BFIND(PPC.PPFE,"DESCR") THEN DATA.TL.TL.TL.HD;
 3   ELSE [%DATA.HD%];
 4   CLOSE;
 5   END;
 6
 7   FUNCTION INITR;
 8   IF BFIND(PPC.PPFE,"EFF") OR BFIND(PPC.PPFE,"OP") THEN [%DATA.HD%];
 9   ELSE DATA.TL.TL.TL.HD;
10   CLOSE;
11   END;
12
13   FUNCTION GOAL; DATA.TL.TL.TL.HD; END;
14
15   FUNCTION PROCESS; DATA.TL.TL.HD::NIL; END;
16
17   FUNCTION ENVIR;
18   IF ("FINITE",ISIN(ADJUNCT.HD.TL.HD)) THEN ADJUNCT.HD;
19   ELSE ADJUNCT.TL.HD;
20   CLOSE;
21   END;
22
23   FUNCTION APPENDIX;
24   VARS ST;
25   WHICH(ADJUNCT, LAMBDA X;
26     IF ("NONFIN",ISIN(X.TL.HD)) OR ("ADVERB",ISIN(X.TL.HD)) THEN 1;
27     ELSE 0;
28     CLOSE;
29    END);
30   ->ST;
31   IF ST.GIVEN THEN ST.HD; ELSE NIL; CLOSE;
32   END;
33
34
```

```
[CLAUSE RULES]                          [ 20.47 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974


 1   VARS CLFTRRS;
 2   [
 3    [CLAUSE     [+PROCESS IFF [SIMPLE]]]
 4    [EFF        [+GOAL  IFF [-GINTRANS USTANDG]]
 5                [+ACTOR IFF [-USTANDA]]]
 6    [DESCR      [+ACTOR IFF [-USTANDA]]]
 7    [OP         [+INITR = +SUBJECT IFF [-USTANDI]]
 8                [ACTOR = SUBJECT IFF [EFF] AND [-USTANDA]] ]
 9    [MID        [+INITR = +SUBJECT = ACTOR IFF [-USTANDI]]
10                [INITR = GOAL IFF [-USTANDI USTANDG DESCR]]]
11    [RECEPTIV   [+SUBJECT = GOAL IFF [EFF]]
12                [+SUBJECT = ACTOR IFF [DESCR]]
13                [+PASSIVE]]
14    [INDIC      [+FINITE]]
15    [INTERROG   [+WHOC]]
16    [MODAL      [+MODAL]]
17    [MODALFUT   [+MODALFUT=MODAL]]
18    [MODALPOS   [+MODALPOS=MODAL]]
19    [PAST       [+PAST]]
20    [FINITE     [+FINITE]]
21    [BOUND      [+BINDER]]
22    [IFBOUND    [+IFB=BINDER]]
23    [THOBOUND   [+THO=BINDER]]
24    [PREPREL    [+PREPREL IFF [EXPLIC]]]
25    [NOMREL     [+REL IFF [EXPLIC]]]
26    [DANGLING   [+DANGLINGP]]
27    [GOALREL    [REL=GOAL IFF [EXPLIC] AND [-USTANDG]]]
28    [INFIN      [+INFIN=PROCESS]]
29    [PARTICIPL  [+PARTICIPLE = PROCESS]]
30    [ING        [+ING = PARTICIPL]]
31    [PERFECTIV  [+PERFECT]]
32    [IMMINENT   [+IMMINENT]
33                [+INFIN=PROCESS]]
34    [ENVIRADJ   [+ENVIR = +CLAUSE]]
35    [IFADJ      [+HYPOTH=ENVIR]]
36    [THOADJ     [+CONCESS=ENVIR]]
37    [APPADJ     [+APPENDIX]]
38    [ACCOMPADJ  [+WITHOBJ = APPENDIX]]
39    [METHODADJ  [+BYOBJ=APPENDIX]]
40    [TIMEADJ    [+TIME = APPENDIX]]
41    [TFIRST     [+FIRST = TIME]]
42    [TYET       [+YET = TIME]]
43    [PRADJ      [+RESTRICT = +ADVERB]]
44    [PRJUST     [+JUST = ADVERB]]
45    [PRYET      [+YET = ADVERB]]
46    [NEGATIVE   [+NEG = ADVERB]]
47   ];
48    ->CLFTRRS;
49
50
51
52   VARS CLSBRS1 CLSBRS2;
53   [
54    [+COMMA IFF [ENVIR]]
55    [+EN = PROCESS IFF [PASSIVE PERFECT]]
56    [+POSTVERB = ACTOR IFF [ACTOR] AND [-ACTOR=SUBJECT]
```

```
                                AND [-ACTOR = REL]]
 [+POSTVERB = GOAL IFF [GOAL] AND [-GOAL=SUBJECT] AND [-GOAL=REL]]
 [POSTVERB = DANGLINGP IFF [POSTVERB] AND [DANGLINGP]]
]
->CLSBRS1;

[
 [ENVIR > [PREPREL REL MFOC BINDER] > SUBJECT > PROCESS >
  POSTVERB > APPENDIX]
 [FINITE = PAST = MFOC = [MODAL PERFECT PASSIVE IMMINENT PROCESS]]
 [MODAL > PERFECT > PASSIVE > IMMINENT > PROCESS]
 [FINITE > RESTRICT]
 [ENVIR > COMMA]
]
->CLSBRS2;

VARS CLFURRS;
[
 [ADVERB       [+ADVERB] [+WORD]]
 [APPENDIX     [+PREPG IFF [ BYOBJ] OR [ WITHOBJ]]
               [+ADVERB IFF [ TIME]]
               [+WORD IFF [ TIME]]]
 [BINDER       [+WORD] [+BINDER]]
 [BYOBJ        [+BY]]
 [CLAUSE       [+CLAUSE]]
 [COMMA        [+WORD] [+LINK] [+COMMA]]
 [CONCESS      [+THOBOUND]]
 [DANGLINGP    [+DANGLINGP]]
 [EN           [+PARTICIPLE] [+PASTPART]]
 [ENVIR        [+DEPENDENT] [+FINITE] [+BOUND]]
 [FINITE       [+TENSED]
               [+REMOTE IFF [PAST]] [+PRESNT IFF [-PAST]]]
 [FIRST        [+FIRST]]
 [HYPOTH       [+IFBOUND]]
 [IFB          [+IF]]
 [IMMINENT     [+VG] [+GRAMMATICAL] [+BEGOING]]
 [INFIN        [+INFINITIVE] [+TO]]
 [ING          [+PARTICIPLE] [+PRESPART]]
 [NEG          [+NEGATIVE]]
 [N1           [+ 1]]
 [N2           [+ 2]]
 [N3           [+ 3]]
 [MODAL        [+VG] [+GRAMMATICAL] [+MODAL]]
 [MODALFUT     [+WILL]]
 [MODALPOS     [+CAN]]
 [PASSIVE      [+VG] [+GRAMMTICAL] [+ASPECTUAL] [+BE]]
 [PERFECT      [+VG] [+GRAMMATICAL] [+ASPECTUAL] [+HAVE]]
 [PLURAL       [+PL]]
 [POSTVERB     [+OBJECT] [+NG]]
 [PREPREL      [+PREPG IFF [-DANGLINGP]]
               [+OF IFF [-DANGLINGP]]
               [+NG IFF [DANGLINGP]]
               [+RELATIVE]]
 [JUST         [+JUST]]
 [PROCESS      [+VG] [+LEXICAL]]
 [REL          [+NG] [+RELATIVE]]
 [SINGULAR     [+SNG]]
 [SUBJECT      [+NG] [+SUBJECT]]
 [THO          [+ALTHOUGH]]
 [WITHOBJ      [+WITH]]
```

```
117     [YET            [+YET]]
118    ]
119    ->CLFURRS;
120
```

```
[CONJUNCT TABLE]                          [ 20.33 11 MAY 1974]
DTRACK 217     CREATED    20.30    11   5 1974


  1    [%-4,1%]-> "COMPLETE".MEANING;
  2    [%0,1%]->"START".MEANING;
  3    [%0,1%]->"BEGIN".MEANING;
  4    [%0,1%]->"TAKE".MEANING;
  5    [%-2,1%]->"BLOCK".MEANING;
  6    [%0,2%]->"THREATEN".MEANING;
  7    [%0,3%]->"FORK".MEANING;
  8    [%-4,0%]->"WIN".MEANING;
  9    [%-4,0%]->"DRAW".MEANING;
 10
 11    FUNCTION PROVAL V; V.MEANING.TL.HD; END;
 12
 13    FUNCTION RETROVAL V; V.MEANING.HD; END;
 14
```

```
[DESIGN0]                          [ 20.48 11 MAY 1974]
DTRACK 217     CREATED    20.45    11   5 1974


    VARS ASSESSLIST;
    FUNCTION ASSESS MOVE => MOVE;
    VARS X CPL OCPL ST HOW;
    MOVE.HD->CPL; OPPONENT(CPL)->OCPL;

    COMMENT 'CONVERT NAME OF SQUARE TO SQUARE';
     IF MOVE.TL.HD.ISNUMBER THEN
     MOVE.TL.HD.COORDS.BOARDV->MOVE.TL.HD; CLOSE;
     NIL->MOVE.TL.TL;

    COMMENT 'EVALUATE THIS MOVE.';
    IF .MOVENO=1 THEN
     (MOVE<>([%CHOICEOF([START BEGIN]),[%HD(£GAME)%],
        "TAKE", [%MOVE.TL.HD%]%]))->MOVE;
    EXIT;
    APPTESTLIST(ASSESSLIST,
     LAMBDA F;
       CPL; IF NOT(BOOLOR(F="WIN",F="DRAW")) THEN OCPL; CLOSE;
       IF SOMEHOW(F) THEN ->HOW; ELSE 0; EXIT;
      F.FNPROPS.HD->F;
      IF NOT(MOVE.TL.HD,ISIN(HOW)) THEN 0; EXIT;
      IF F="WIN" THEN
       WHICH(£LINES,
        LAMBDA X; USEFUL(X,CPL)+(MOVE.TL.HD,ISIN(£X))=2; END)
       ->ST;
       [%F,[%HD(£GAME)%],"COMPLETE",ST%];
      ELSE
       WHICH(£LINES,
        LAMBDA X; (USEFUL(X,OCPL)+(MOVE.TL.HD,ISIN(£X)))=2; END)
       ->ST;
       IF (F,ISIN([%"THREATEN","FORK"%])) THEN OCPL;
       ELSE HD(£GAME); CLOSE;
       ->X;
       IF ST.GIVEN THEN
        IF F="BLOCK" THEN
       IF HOW.LENGTH=1 THEN [%F,ST%];
       ELSE [%F,ST,"TAKE", [%MOVE.TL.HD%]%]; CLOSE;
       ELSE [%"BLOCK",ST,F,[%X%]%];
        CLOSE;
       ELSEIF HOW.LENGTH=1 THEN [%F,[%X%]%];
       ELSE [%F,[%X%],"TAKE", [%MOVE.TL.HD%]%];
       CLOSE;
      CLOSE;
      ->ST;
      MOVE<>ST->MOVE; 1;
     END);

    L1:
    IF MOVE.TL.TL.GIVEN THEN INTERSECT([DRAW WIN],MOVE).GIVEN->LAST; EXIT;
    (MOVE<>[%"TAKE",[%MOVE.TL.HD%]%])->MOVE;
    END;

                                298
```

```
[DESIGN1]                          [ 20.49 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974

 1   FUNCTION DESIGN1 => OUT;
 2   IF DESIGN1.FNPROPS.TL.GIVEN THEN
 3     DESIGN1.FNPROPS.TL.DEST->DESIGN1.FNPROPS.TL ->OUT;
 4   CLOSE;
 5   VARS MOVE1 ST ST1 ST2 HMAJVB HMIVB
 6        MAJVB1 MAJVB2 MAJGL1 MAJGL2 MIVB1 MIVB2 MIGL1 MIGL2 MAJCLCNT;
 7   0->MAJCLCNT; NIL->OUT;
 8   LOOP:
 9    IF MAJCLCNT=4 THEN
10     LOOPIF OUT.HD.ISWORD THEN OUT.TL->OUT; CLOSE;
11     OUT.DEST->OUT; -->REST; NIL->PPC.PPSELF.HD.TL.HD.CONTENTS.TL;
12     3->MAJCLCNT; PPC.PPSELF.TL->PPC.PPSELF; GOTO REVISE;
13    ELSEIF REST.NULL THEN;
14     GOTO REVISE;
15    CLOSE;
16    REST.DEST->REST; .ASSESS->MOVE1; MOVE1-->ROOT.PPSELF;
17    NIMMT(MOVE1.HD,MOVE1.TL.HD);
18    MOVE1.TL.TL.HD->MAJVB1; MOVE1.TL.TL.TL.HD->MAJGL1;
19   IF MAJGL1.GIVEN THEN MAJGL1.HD->MAJGL1; CLOSE;
20    MOVE1.HD->CPL;
21    IF MOVE1.LENGTH>4 THEN
22     MOVE1.TL.TL.TL.TL.HD->MIVB1; MOVE1.TL.TL.TL.TL.TL.HD->MIGL1;
23    IF MIGL1.GIVEN THEN MIGL1.HD->MIGL1; CLOSE;
24    ELSE NIL->MIVB1; NIL->MIVB2;
25    CLOSE;
26
27    COMMENT 'MAKE A NOTE OF COORD OR SUBORD CLAUSE STRUCTURE';
28    IF MIVB1.GIVEN AND MIVB1.PROVAL>1 THEN 2+MAJCLCNT;
29    ELSE 1+MAJCLCNT;
30    CLOSE;->MAJCLCNT;
31    MOVE1-->OUT;
32
33   COMMENT 'TEST FOR FORK.';
34    IF NOT(MIVB1=NIL) THEN MIVB1.PROVAL; ELSE MAJVB1.PROVAL; CLOSE;
35    ->ST;
36    IF ST=3 AND REST.GIVEN THEN; GOTO REVISE; CLOSE;
37
38    COMMENT 'TEST FOR HYPOTHETICAL REQUIRED.';
39    IF (MOVE1.HD,EQUAL(PROTEUS)) THEN; GOTO L1; CLOSE;
40    NIL->MOVE1.TL.HD.CONTENTS.TL;
41    .PLAN->ST1; NIMMT(CPL,MOVE1.TL.HD);
42    ST1.TL.TL.HD->HMAJVB;
43    IF ST1.TL.TL.TL.TL.GIVEN THEN
44     ST1.TL.TL.TL.TL.HD->HMIVB;
45    ELSE NIL->HMIVB;
46    CLOSE;
47    IF (HMAJVB,EQUAL(MAJVB1)) THEN NIL->ST1; GOTO L1; CLOSE;
48    IF REST.NULL THEN;
49    ELSEIF HMAJVB.RETROVAL= (0-4) THEN;
50    ELSEIF HMAJVB.RETROVAL= 0 THEN
51     IF HMIVB.GIVEN THEN HMIVB; ELSE HMAJVB; CLOSE;
52     .PROVAL->ST;
53     IF MIVB1.GIVEN THEN MIVB1; ELSE MAJVB1; CLOSE;
54     .PROVAL->ST2;
55     IF ST2=<ST THEN; GOTO L1; CLOSE;
56    ELSEIF HMAJVB.RETROVAL<0 THEN
```

```
 57      WHICH(£LINES,USEFUL(%OPPONENT(CPL)%))->ST;
 58      IF SOME(ST,LAMBDA X; (ST1.TL.HD,NOT(ISIN(£X))); END) THEN;
 59       GOTO L1;
 60      CLOSE;
 61      ASSESS(REST.HD)->ST;
 62      IF ST.TL.TL.HD.RETROVAL> (0-4) THEN; GOTO L1; CLOSE;
 63     CLOSE;
 64
 65     COMMENT 'SO WE DO NEED A HYPOTHETICAL.';
 66     IF OUT.LENGTH>1 THEN;
 67      OUT.DEST->OUT; -->REST; NIL->PPC.PPSELF.HD.TL.HD.CONTENTS.TL;
 68      PPC.PPSELF.TL->PPC.PPSELF; GOTO REVISE;
 69     CLOSE;
 70    IF HMIVB=NIL THEN
 71      [%(("IF"::ST1)::MOVE1)%]->OUT;
 72     ELSE [%MOVE1,"BUT",ST1,"IF"%]->OUT;
 73     CLOSE;
 74     ","-->OUT;
 75
 76    L1: IF REST.NULL THEN
 77     IF LENGTH(OUT)=1 AND MAJVB1.RETROVAL<0 THEN
 78      [%OUT.HD,",","HOWEVER"%]->OUT;
 79     CLOSE;
 80     GOTO REVISE;
 81    CLOSE;
 82
 83
 84     REST.HD.ASSESS->REST.HD;
 85     REST.HD.TL.TL->ST;
 86     ST.HD->MAJVB2; ST.TL->ST; ST.HD->MAJGL2; ST.TL->ST;
 87     IF ST.GIVEN THEN ST.HD->MIVB2; ST.TL.HD->MIGL2;
 88     ELSE NIL->MIVB2;
 89     CLOSE;
 90     IF (OUT.LENGTH=1) AND (MAJVB1.RETROVAL<0)
 91      AND (MAJVB2.RETROVAL>=0) THEN
 92      [%OUT.HD,",","HOWEVER"%]->OUT;
 93     CLOSE;
 94
 95     MAJVB1.PROVAL+MAJVB2.RETROVAL->ST;
 96     IF ST>0 THEN "," -->OUT; GOTO LOOP; CLOSE;
 97     IF ST=0 THEN
 98      IF OUT.LENGTH=1 THEN "BUT"-->OUT; GOTO LOOP;
 99      ELSE MOVE1-->REST; OUT.TL->OUT;
100       NIL->PPC.PPSELF.HD.TL.HD.CONTENTS.TL; PPC.PPSELF.TL->PPC.PPSELF;
101      GOTO REVISE;
102      CLOSE;
103     ELSEIF ST=(0-3) OR ST=(0-2) THEN
104       IF MAJVB1.RETROVAL=(0-2) THEN
105         COMMENT 'A HOPELESS ATTEMPT TO BLOCK A WIN.';
106         IF OUT.LENGTH=1 THEN
107          REST.HD.ASSESS->REST.HD;
108           [%(("ALTHOUGH"::OUT.HD)::REST.HD)%]->OUT;
109           REST.DEST->REST;-->PPC.PPSELF;
110           GOTO LOOP;
111         ELSE OUT.HD-->REST;  OUT.TL->OUT;
112           NIL->PPC.PPSELF.HD.TL.HD.CONTENTS.TL;
113           PPC.PPSELF.TL->PPC.PPSELF; GOTO REVISE;
114         CLOSE;
115       ELSE; COMMENT 'AN UNOPPOSED WIN.';
116         ("SO"::("AND"::OUT))->OUT; GOTO LOOP;
```

```
117       CLOSE;
118     CLOSE;
119
120    REVISE:
121     COMMENT '1. REPLACE FINAL "," BY "AND ,". `;
122     IF OUT.HD.ISWORD THEN OUT.TL->OUT; GOTO REVISE; CLOSE;
123     IF OUT.TL.GIVEN AND OUT.TL.HD="," THEN
124      IF OUT.TL.TL.GIVEN AND NOT(OUT.TL.TL.HD="HOWEVER") THEN
125       OUT.HD->ST; "AND"->OUT.HD; ST-->OUT;
126      CLOSE;
127     CLOSE;
128
129     COMMENT '2. ADD FINAL "." `;
130     "."-->OUT;
131
132     COMMENT '3. DELETE INITIAL "HOWEVER" IF "BUT" OCCURS.`;
133     IF ("BUT",ISIN(OUT)) AND ("HOWEVER",ISIN(OUT)) THEN
134      OUT.REV.TL.TL.REV->OUT;
135     CLOSE;
136
137     COMMENT '4. UNDO GEDANKEN MOVES.`;
138     APPLIST(PPC.PPSELF,LAMBDA X; NIL->X.TL.HD.CONTENTS.TL: END);
139    PPC.PPSELF.REV->PPC.PPSELF;
140    END;
141
```

```
[DETERMIN]                          [ 20.47 11 MAY 1974]
DTRACK 217     CREATED    20.44     11  5 1974


    1    FUNCTION NUMDET REF XEQCL;
    2    IF CHECKMEN(REF,XEQCL) THEN 1;
    3    ELSEIF NUMBEROF(REF)<NUMBEROF(XEQCL) AND NUMBEROF(REF)>1 THEN 1;
    4    ELSE 0; RETURN;
    5    CLOSE;
    6    "NUMBERED"-->FLIST;
    7    END;
    8
    9    FUNCTION DEFQ REF;
   10    VARS REFNO;
   11    REF.LENGTH->REFNO;
   12    IF ("NOTDET",ISIN(FLIST)) OR ("DETERMINED",ISIN(FLIST)) THEN EXIT;
   13    IF REFNO=NUMBEROF(XEQCL) THEN "DEF";
   14    ELSE "INDEF";
   15    CLOSE;
   16    -->FLIST;
   17    END;
   18
```

```
[ENTITY]                    [ 20.33 11 MAY 1974]
DTRACK 217    CREATED   20.30   11  5 1974


 1   VARS CONSENTITY TYPE NNAME CONTENTS HYPOTH;
 2   RECORDFNS("ENTITY",[0 0 0 0])->HYPOTH->CONTENTS->NNAME->TYPE;
 3   .ERASE; ->CONSENTITY;
 4
 5   MACRO EVARS;
 6   VARS X Y;
 7   .ITEMREAD->X; .ITEMREAD->Y;
 8   [%"VARS",X,";"%]==>;
 9   ([CONSENTITY(]<>[%[%Y.VALOF%]%]<>[,"]<>
10      [%X%]<>[",NIL,NIL)->]<>[%X,";"%])==>;
11   [%X,"-->","CONTENTS","(",Y,")",";"%]==>;
12   END;
13
14   VARS ENTITY; CONSENTITY(NIL,"ENTITY",NIL,NIL)->ENTITY;
15   EVARS GAME ENTITY;
16   EVARS HISTORY GAME;
17   EVARS PLAYERS ENTITY;
18   EVARS PERSON PLAYERS;
19   EVARS ACD PERSON; "ACD"--> £ACD;
20   EVARS HCL PERSON; "HCL"--> £HCL;
21   EVARS STEVE PERSON; "STEVE"--> £STEVE;
22   EVARS DAN PERSON; "DAN"--> £DAN;
23   EVARS BOB PERSON; "BOB"--> £BOB;
24   EVARS CLARIBEL PERSON; "CLARIBEL"--> £CLARIBEL;
25
26   EVARS PROGRAM PLAYERS;
27   EVARS PROTEUS PROGRAM; "PROTEUS"--> £PROTEUS;
28
29   EVARS LOCATION ENTITY;
30
31   EVARS SQUARES LOCATION;
32   EVARS CORNERS SQUARES;
33   [BOARD]↑↑↑
34    [%SQUARE 1,SQUARE 3,SQUARE 7,SQUARE 9%]-> £CORNERS;
35    [%APPLIST(£CORNERS,IDENTFN),SQUARE 5,SQUARE 2,SQUARE 4,SQUARE 6,
36      SQUARE 8%] -> £SQUARES;
37   £SQUARES-> £BOARD;
38   EVARS LINES LOCATION;
39
40   EVARS C2 LINES; [%SQUARE 2,SQUARE 5,SQUARE 8%]-> £C2;
41   EVARS R2 LINES; [%SQUARE 4,SQUARE 5,SQUARE 6%]-> £R2;
42   EVARS EDGES LINES;
43   EVARS DIAGONALS LINES;
44   EVARS D1 DIAGONALS; [%SQUARE 3,SQUARE 5,SQUARE 7%]-> £D1;
45   EVARS D2 DIAGONALS; [%SQUARE 1,SQUARE 5,SQUARE 9%]-> £D2;
46   EVARS C1 EDGES; [%SQUARE 1,SQUARE 4,SQUARE 7%]-> £C1;
47   EVARS C3 EDGES; [%SQUARE 3,SQUARE 6,SQUARE 9%]-> £C3;
48   EVARS R1 EDGES; [%SQUARE 1,SQUARE 2,SQUARE 3%]-> £R1;
49   EVARS R3 EDGES; [%SQUARE 7,SQUARE 8,SQUARE 9%]-> £R3;
50    [%R1,R3,C1,C3%]-> £EDGES;
51    [%D1,D2%]-> £DIAGONALS;
52    (£EDGES<> £DIAGONALS<>[%C2,R2%])-> £LINES;
53
54
```

```
[EQCL]                          [ 20.33 11 MAY 1974]
DTRACK 217    CREATED    20.30    11  5 1974


 1   FUNCTION SMAP Q  T; BOARDV(T(COORDS(Q.HD)));  END;
 2   FUNCTION LMAP Q T;
 3   COMMENT 'TRANSFORMS Q BY T AND IDENTIFIES THE RESULT.';
 4   MAPLIST(Q,LAMBDA X; X.CONTENTS.HD.COORDS.T.COORDS2N; END)->Q;
 5   IF (Q.HD>Q.TL.HD) THEN REV(Q)->Q; CLOSE;
 6   (Q.TL.HD-Q.HD)->T;
 7   IF T=1 THEN GET(INTOF(((Q.HD-1)/3)+1),[R1 R2 R3]);
 8   ELSEIF T=3 THEN GET(Q.HD,[C1 C2 C3]);
 9   ELSE GET(INTOF(((Q.HD-1)/2)+1),[D2 D1]);
10   CLOSE; .VALOF;
11   END;
12
13   FUNCTION NONMAP XEQCL COMPLEQCL FNLIST;
14   VARS F;
15   LOOPIF FNLIST.GIVEN THEN FNLIST.DEST->FNLIST->F;
16    IF ALL(XEQCL,F) AND NO(COMPLEQCL.ENVIRON,F) THEN
17     F.FNPROPS.HD->F;
18     IF F="END1" THEN "END" ->F; CLOSE;
19     [%F%], 1;
20    EXIT;
21   CLOSE;
22   0;
23   END;
24
25
26   FUNCTION EQCL REFX => L;
27   IF REFX.NULL THEN NIL->L; EXIT;
28   WHICH(EQCL.FNPROPS.TL,
29    LAMBDA X; X.HD,EQUAL(REFX.HD); END)->L;
30   IF L.NULL.NOT THEN L.HD.TL.HD->L; EXIT;REFX->L;
31   NIL->L;
32   REFX.HD-->L; REFX.HD->REFX;
33   IF NOT(REFX,ISA(LOCATION)) OR (REFX,EQUAL(BOARD)) THEN EXIT;
34   VARS T;
35   APPLIST([[-J,I] [-I,-J] [J,-I] [-I,J] [I,-J] [J,I] [-J,-I]],
36    LAMBDA X; ([LAMBDA I J;]<>X<>[; END GOON]).POPVAL->T;
37    IF MATREQ(T) THEN; ELSE EXIT;
38    IF (REFX,ISA(SQUARES)) THEN SMAP(£REFX,T);
39    ELSE LMAP(£REFX,T); CLOSE;
40    -->L; END;);
41   CONDENSE(L)->L;
42   END;
43   LAMBDA X Y; ((Y::(X::NIL))::EQCL.FNPROPS.TL)->EQCL.FNPROPS.TL; END;
44   ->UPDATER(EQCL);
45
46
```

```
1   FUNCTION GETCOND CSR;
2   COMMENT 'RETURNS "CONDITION,1" ELSE 0';
3   LOOPIF CSR.GIVEN THEN
4     IF CSR.HD="IFF" THEN CSR.TL, 1; RETURN; CLOSE;
5     CSR.TL->CSR;
6   CLOSE:
7   0;
8   END;
9
10  FUNCTION TESTCOND COND LIST => LIST SW;
11  COMMENT 'TESTS COND, SETTING SW.';
12  VARS ST F;
13  0->SW;
14  L1: IF COND.NULL THEN EXIT;
15    COND.DEST->COND->ST;
16    IF ST="AND" THEN
17      IF SW=0 THEN RETURN;
18      ELSE 0->SW; GOTO L1;
19      CLOSE;
20    ELSEIF ST="OR" THEN
21      IF SW=1 THEN RETURN;
22      ELSE; GOTO L1;
23      CLOSE;
24    CLOSE;
25
26  COMMENT 'ST IS CONDITION EXPRESSED AS A LIST.'
27          'CHECK IF NEGATIVE.';
28    IF ST.HD="-" THEN ST.TL->ST; NOT;
29    ELSE IDENTFN;
30    CLOSE;
31    ->F;
32
33  COMMENT 'SEE IF THIS IS AN EQUALITY CONDITION.';
34    IF ("=",ISIN(ST)) THEN
35      IF BFETCH(LIST,ST.HD) THEN
36        -->LIST;
37        IF LIST.HD.ISLIST AND (ST.TL.TL.HD,ISIN(LIST.HD)) THEN
38        1->SW;
39        CLOSE;
40        F(SW)->SW; GOTO L1;
41      CLOSE;
42    CLOSE;
43
44  COMMENT 'THE CONDITION IS A PRESENCE/ABSENCE CONDITION.';
45    LOOPIF ST.GIVEN THEN
46      IF BFETCH(LIST,ST.HD) THEN;
47        -->LIST; 1->SW; NIL->ST;
48      ELSE ST.TL->ST;
49      CLOSE;
50    CLOSE;
51    F(SW)->SW;
52    GOTO L1;
53  END;
54
55  FUNCTION PERFORM RULE LIST => LIST;
56  VARS ST ST1;
```

```
57    NIL->ST1;
58    COMMENT 'CARRIES OUT THE RULE, PUTTING RESULT IN LIST.';
59
60    L1: IF RULE.NULL OR (RULE.HD="IFF") THEN
61        IF ST1.GIVEN THEN; GOTO L4; ELSE RETURN; CLOSE;
62      CLOSE;
63      RULE.DEST->RULE->ST;
64      IF ST="+" THEN RULE.DEST->RULE; -->ST1;
65      ELSEIF ST.ISWORD AND BFETCH(LIST,ST) THEN
66        ->ST;
67      L2: IF ST.ISLIST THEN (ST<>ST1)->ST1;
68        ELSE ST-->ST1;
69        CLOSE;
70      ELSE 'ERROR IN PERFORM'.PRSTRING; .POPREADY;
71      CLOSE;
72
73      IF RULE.GIVEN AND RULE.HD="=" THEN;
74     L3: RULE.TL->RULE;
75        IF RULE.HD.ISLIST THEN
76          RULE.DESTPAIR->RULE->PT;
77          LOOPIF PT.GIVEN THEN
78            IF BFETCH(LIST,PT.HD) THEN->ST; NIL->PT; GOTO L2;
79            ELSE PT.TL->PT;
80            CLOSE;
81          CLOSE;
82        CLOSE;
83        IF RULE.GIVEN AND RULE.HD="=" THEN; GOTO L3; CLOSE;
84      ELSE
85     L4: IF ST1.LENGTH>1 THEN ST1-->LIST;
86        ELSEIF NOT(ST1=NIL) THEN ST1.HD-->LIST;
87        CLOSE;
88        NIL->ST1;
89      CLOSE;
90
91    GOTO L1;
92    END;
93
94    FUNCTION REALISE INPUT RULES => OUTPUT;
95    VARS CR CSR ST SW;
96    NIL->OUTPUT;
97    COMMENT 'REALISES THE INPUT ACC. THE RULES AND PUTS IN OUTPUT.';
98
99    L1: IF RULES.NULL THEN; EXIT;
100     RULES.DEST->RULES->CR;
101   L2: COMMENT '1. IS THE REALISATE PRESENT.';
102     IF BFETCH(INPUT,CR.HD) THEN; -->INPUT;
103     ELSE; GOTO L1;
104     CLOSE;
105     CR.TL->CR;
106
107   L3: COMMENT '2. GET NEXT SUB-RULE.';
108     IF CR.GIVEN THEN CR.DEST->CR->CSR; 0->SW;
109     ELSE; GOTO L1;
110     CLOSE;
111
112     COMMENT 'GET AND TEST CONDITION, IF ANY, ON RULE.';
113     IF GETCOND(CSR) THEN ->ST;
114       TESTCOND(ST,INPUT)->SW->INPUT;
115       IF SW=0 THEN; GOTO L3; CLOSE;
116     ELSE 1->SW;
```

```
117       CLOSE;
118
119       COMMENT 'THE CONDITION, IF ANY, IS TRUE, SO DO JOB.';
120       PERFORM(CSR,OUTPUT)->OUTPUT;
121     GOTO L3;
122     END;
123
```

```
[GAMEFNS]                          [ 20.46 11 MAY 1974]
DTRACK 217     CREATED    20.44    11   5 1974


 1    FUNCTION BLOCK PL PL1 => OUT;
 2    VARS ST X; NIL->OUT;
 3    £LINES->ST;
 4    LOOPIF ST.GIVEN THEN ST.DEST->ST->X;
 5      IF (X,USEFUL(PL1)) THEN £X->X;
 6        LOOPIF X.HD.OCCUPIED THEN X.TL->X; CLOSE;
 7        X.HD-->OUT;
 8      CLOSE;
 9    CLOSE;
10    END;
11
12    FUNCTION DRAW PL => OUT;
13    VARS ST X;
14    NIL->OUT; £SQUARES->ST;
15    LOOPIF ST.GIVEN THEN ST.DEST->ST->X;
16      IF X.VACANT THEN NIMMT(PL,X);
17        IF ALL(£LINES,DEAD) THEN X-->OUT; CLOSE;
18        NIL-> £X.TL;
19      CLOSE;
20    CLOSE;
21    END;
22
23    FUNCTION WIN PL => OUT;
24    VARS ST X;
25    NIL->OUT; £LINES->ST;
26    IF .MOVENO<5 THEN EXIT;
27    LOOPIF ST.GIVEN THEN ST.DEST->ST->X;
28      IF (X,USEFUL(PL)) THEN £X->X;
29        LOOPIF X.HD.OCCUPIED THEN X.TL->X; CLOSE;
30        X.HD-->OUT;
31      CLOSE;
32    CLOSE;
33    END;
34
```

```
[HCLHFNS]                          [ 20.33 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974


 1   FUNCTION WHICH XS RY => XXS;
 2   VARS XST; NIL->XXS;
 3   LOOPIF XS.GIVEN THEN XS.DEST->XS->XST;
 4    IF XST.RY THEN XST-->XXS; CLOSE;
 5   CLOSE;
 6   XXS.NCREV->XXS;
 7   END;
 8
 9   FUNCTION NUMBEROF XS; LENGTH(XS); END;
10
11   FUNCTION HOWMANY XS RY => XN;
12   0->XN;
13   LOOPIF XS.GIVEN THEN XS.DEST->XS; (.RY+XN)->XN; CLOSE;
14   END;
15
16   FUNCTION SOME XS RY; HOWMANY(XS,RY)>0; END;
17
18   FUNCTION EVERY XS RY;
19    BOOLAND(XS.GIVEN,HOWMANY(XS,RY)=NUMBEROF(XS)); END;
20
21   FUNCTION ALL XS RY; EVERY(XS,RY); END;
22
23   FUNCTION NO XS RY; IF XS.NULL THEN 1; EXIT; EVERY(XS,:+ RY NOT); END;
24
25
26   FUNCTION ISA Q T;
27   IF NOT(SAMEDATA(Q,T)) THEN 0; EXIT;
28   LOOP:
29    IF EQUAL(Q,T) THEN 1; EXIT;
30    Q.TYPE->Q;
31    IF Q.NULL THEN 0; ELSE Q.HD->Q; GOTO LOOP; CLOSE;
32   END;
33
34   FUNCTION OCCUPIED Q;
35     IF (Q,ISA(LINES)) THEN ALL(£Q,OCCUPIED); EXIT;
36     LENGTH(£Q)=3;
37   END;
38   FUNCTION VACANT X;
39    IF (X,ISA(SQUARES)) THEN LENGTH(£X)=1; EXIT;
40    IF (X,ISA(LINES)) THEN ALL(£X,VACANT); EXIT;
41    EVERY(£LINES,VACANT);
42   END;
43
44   FUNCTION EMPTY; .VACANT; END;
45   FUNCTION FREE; .VACANT; END;
46
47   FUNCTION USEFUL XLINE XPLAYER;
48   VARS ST9 ST8 ST7; 0->ST7; 0->ST8; £XLINE->XLINE;
49   LOOPIF XLINE.GIVEN THEN XLINE.DEST->XLINE->ST9;
50    IF ST9.VACANT THEN ST7+1->ST7;
51    ELSE ((£ST9.TL.HD,EQUAL(XPLAYER))+ST8)->ST8; CLOSE;
52   CLOSE;
53   BOOLAND(ST7=1,ST8=2);
54   END;
55
56   FUNCTION OPEN XLINE; HOWMANY(£XLINE,VACANT)=2; END;
```

```
57
58   FUNCTION DEAD XLINE;
59
60   VARS ST7 ST8 ST9; 0->ST7; 0->ST8;
61   IF (XLINE,NOT(ISA(LINES))) THEN 0; EXIT;
62   £XLINE->XLINE;
63   LOOPIF XLINE.GIVEN THEN XLINE.DEST->XLINE->ST9;
64    IF ST9.OCCUPIED THEN £ST9.TL.HD->ST9;
65     IF (ST9,EQUAL(CPL)) THEN ST7+1->ST7;
66     ELSE ST8+1->ST8; CLOSE;
67    CLOSE;
68   CLOSE;
69   BOOLAND(ST7>0,ST8>0);
70   END;
71
```

```
[LOCDESCR]                    [ 20.48 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974


 1   FUNCTION LOCDESCR PPC WHDEPTH => OK;
 2   VARS DATA REF SAVEFTS ST XEQCL XTYPE;
 3   PPC.PPSELF->ST; NIL->DATA; PPC.PPDAD.PPFE.COPYLIST->SAVEFTS;
 4   ST.DEST->ST ->XEQCL;
 5   ST.DEST->ST ->XTYPE;
 6   ST.DEST->ST ->REF;
 7   IF NOT(COMPLEQCL,ISLIST) THEN
 8     COMPLEMENT(XEQCL,£XTYPE)->COMPLEQCL;
 9   CLOSE;
10   IF ("DANGLINGP",ISIN(PPC.PPDAD.PPFE)) OR
11      ("PREPREL",ISIN(PPC.PPDAD.PPDAD.PPFE)) THEN
12     GOTO MOD9;
13   CLOSE;
14   REF-->CANLIST;
15
16   MOD1:
17     COMMENT 'TRY DESCRIPTION OF TAKEN SQUARE BY WHEN TAKEN.';
18     IF (REF.HD,ISA(SQUARES)) AND NUMBEROF(REF)=1 AND REF.HD.OCCUPIED
19       THEN REF->PPC.PPSELF; LOCD1(PPC)->OK;
20       IF OK THEN
21         ("QUALIFIED"::("CLAUSEQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
22       EXIT;
23       NIL->PPC.PPSELF;
24     CLOSE;
25
26   MOD2:
27     COMMENT 'TRY DESCRIPTION AS "...OTHER...".';
28     REF->PPC.PPSELF; LOCD2(PPC)->OK;
29     IF OK THEN EXIT;
30
31   MOD3:
32     COMMENT 'TRY DESCRIPTION AS "...SOMEONES...".';
33     COMPLEMENT(XEQCL,£XTYPE)->COMPLEQCL;
34     IF LOCD3(XEQCL,COMPLEQCL,PPC) THEN 1->OK; EXIT;
35
36   MOD4:
37     COMMENT 'ATTEMPT DESCRIPTION AS "...EMPTY...".';
38     LOCD4(PPC)->OK;
39     IF OK THEN EXIT;
40
41   MOD5:
42     COMMENT 'TRY TO DESCRIBE IN RELATION TO IDENTIFIABLE POINT.';
43     LOCD5(REF,XEQCL,PPC)->OK;
44     IF OK THEN EXIT;
45
46   MOD6:
47     COMMENT 'SEE IF THE REFERENT IS THE MIDDLE OF THE BOARD.';
48     LOCD6(REF,PPC)->OK;
49     IF OK THEN EXIT;
50
51   MOD7:
52     COMMENT 'IF REFERENT IS A SQUARE, REFER BY FLANKING ENVIRON.';
53     IF (REF.HD,ISA(SQUARES)) THEN; GOTO MOD9; CLOSE;
54     COMMENT 'TRY IDENTIFYING THE LINE BY THE LAST TWO MOVES.';
55     LOCD7(REF,PPC)->OK;
56     IF OK THEN EXIT;
```

```
57
58    MOD8:
59      COMMENT 'IDENTIFY A LINE BY REFERRING TO THE MIDDLE.';
60      LOCD8(REF,PPC)->OK;
61      IF OK THEN EXIT;
62
63    MOD9:
64      COMMENT 'IDENTIFY REF BY WHAT FLANKS IT.';
65      LOCD9(REF,PPC)->OK;
66      IF OK THEN EXIT;
67      SAVEFTS.COPYLIST->PPC.PPDAD.PPFE;
68    MOD10:
69      COMMENT 'AS A LAST RESORT PRODUCE A REFERENCE IN TERMS OF'
70              'A CROSS BEARING.';
71      IF WHDEPTH>1 THEN 0->OK; NIL->PPC.PPDAD.PPSON; EXIT;
72      LOCD10(XEQCL,COMPLEQCL,PPC)->OK;
73      PPC.PPBRO->PPC.PPDAD.PPSON;
74      IF OK THEN EXIT;
75
76    'ERROR IN LOCDESCR - FAILURE OF LOCD10.'.PRSTRING; .POPREADY;
77
78    END;
79
```

```
1   FUNCTION LOCD1 PPC => OK;
2   COMMENT 'DEFINES AN OCCUPIED SQUARE BY WHEN TAKEN, IF POSSIBLE.';
3   VARS ST ST1 REF;
4   PPC.PPSELF->REF;
5
6   COMMENT '1 FIND WHEN TAKEN.';
7   £HD(REF).TL.TL.HD->ST;
8   IF ST>(.MOVENO-3) OR ST<3 THEN 1->OK;
9   ELSE 0->OK; RETURN;
10  CLOSE;
11
12  COMMENT '2 DEFINE THE RELATIVE CLAUSE, CONTENT & FEATURES.';
13  "CLAUSE"->PPC.PPUNIT;
14  [%£HD(REF).TL.HD,REF.HD,"TAKE",REF%]->PPC.PPSELF;
15  [DEPENDENT FINITE RELATIVE NOMREL GOALREL].COPYLIST->PPC.PPFE;
16
17  COMMENT '3. DECIDE WHETHER TO SUPPRESS THE RELATIVE.';
18  IF CHOICEOF([0 1]) THEN "EXPLIC"; ELSE "USTANDG"; CLOSE;
19  -->PPC.PPFE;
20
21  COMMENT '4. DECIDE TIME ADVERB AND ASPECT.';
22  IF ST>(.MOVENO-3) THEN [PRADJ PRJUST PERFECTIVE].COPYLIST;
23  ELSE [APPADJ TIMEADJ TFIRST].COPYLIST;
24  CLOSE;
25  (<>PPC.PPFE)->PPC.PPFE;
26
27  COMMENT '5. MAKE CLAUSE.';
28  MAKES(PPC.PPSELF,PPC.PPFE,NIL,PPC)->OK;
29  IF NOT(OK) THEN NIL->PPC.PPSON; CLOSE;
30  END;
31
32
33  FUNCTION LOCD2 PPC => OK;
34  COMMENT 'DESCRIBES AS "...OTHER...", IF POSSIBLE.';
35  VARS ST;
36  IF NOT("DE",ISIN(PPC.PPDAD.PPFE)) AND
37     NOT(CHECKDE(PPC.PPSELF,XEQCL)) THEN 0->OK; RETURN;
38  CLOSE;
39  IF WHDEPTH>=2 THEN 0->OK; EXIT;
40  [%"OTHER"%]->PPC.PPSELF;
41  [ADJG SIMPLE].COPYLIST->PPC.PPFE;
42  COMMENT '"OTHER" CAN DO WITHOUT PROSTRING PRONOUN AS HEAD.';
43  1->OK;
44  PPC.PPDAD.PPFE->ST; "MODIFIED"-->ST;
45  IF CHOICEOF([1 0 1]) AND ("PROSTRING",ISIN(ST)) THEN
46    LISTDELETE(ST,[PRONOUN PROSTRING])->ST;
47  CLOSE;
48  ST->PPC.PPDAD.PPFE;
49  MAKEADJG(PPC);
50  END;
51
52
53  FUNCTION LOCD3 XEQCL COMPLEQCL PPC => OK;
54  VARS ST F;
55  XEQCL.HD->ST;
56  IF NOT(ST,ISA(LINES)) THEN 0->OK; EXIT;
```

```
57
58    COMMENT 'TRY TO SAY THAT THE LINE BELONGS TO SOMEONE.';
59    WHICH([%CPL,OPPONENT(CPL)%],
60      LAMBDA X; USEFUL(ST,X); END) ->ST;
61    IF ST.NULL THEN 0->OK; EXIT;
62    ST.HD->ST;
63    LAMBDA X; USEFUL(X,ST); END->F;
64    IF ALL(XEQCL,F) AND NO(COMPLEQCL,F) THEN 1->OK;
65    ELSE 0->OK; RETURN;
66    CLOSE;
67
68    COMMENT 'SPECIFY WHETHER PRONOUN OR DETERMINER.';
69    PPC.PPDAD.PPFE->F;
70    NIL->PPC.PPFE; "POSSESS"-->F;;
71    IF ("PROSTRING",ISIN(F)) THEN
72      ("PROREF"::(LISTDELETE(F,"PROSTRING")))->F;
73    CLOSE;
74    F->PPC.PPDAD.PPFE;
75
76    COMMENT 'NOW MAKE DETERMINER OR PRONOUN.';
77    [POSSESS SG AN].COPYLIST->FLIST; [%ST%].PERSONQ; [%ST%].ANIMATEQ;
78    FLIST->PPC.PPFE;
79    IF ("PRONOUN",ISIN(F)) THEN
80      "PRONOUN"-->PPC.PPFE; MAKEWORD(PPC); RETURN;
81    ELSEIF PRONOUN([%ST%]) THEN
82      "ARTICLE"-->PPC.PPFE; MAKEWORD(PPC); RETURN;
83    CLOSE;
84    COMMENT 'THE DETERMINER IS A NOUN.';
85    "DET"->PPC.PPUNIT;
86    ADDTO(CONSNODE(NIL,[NG GENITIVE],[%ST%],PPC,NIL,NIL));
87    MAKENG(PPC.PPSON); .ERASE;
88    END;
89
90
91    FUNCTION LOCD4 PPC => OK;
92    COMMENT 'TRY DESCRIPTION AS "...EMPTY...".';
93    IF NONMAP(XEQCL,COMPLEQCL,[%CHOICEOF([%VACANT, EMPTY, FREE%])%]) THEN
94      ->PPC.PPSELF; 1->OK;
95      [ADJG SIMPLE].COPYLIST->PPC.PPFE; MAKEADJG(PPC);
96      NIL->PPC.PPDAD.PPSON; "MODIFIED"-->PPC.PPDAD.PPFE;
97    ELSE 0->OK;
98    CLOSE;
99    END;
100
101
102
103   FUNCTION LOCD5 REF XEQCL PPC => OK;
104   COMMENT 'TRY TO DESCRIBE IN RELATION TO SOME IDENTIFIABLE POINT.';
105   VARS PT SAVE ST ST1 ST2; NIL->ST2;
106
107   COMMENT '1. SEE IF LRRP HAS DEFINED SUCH A POINT ALREADY.';
108   L1:
109     LRRP.FNPROPS.TL->ST;
110     IF ST.NULL THEN; GOTO L2; CLOSE;
111     WHICH(ST,LAMBDA X; REF,ISIN(X); END)->ST1;
112     IF ST1.NULL THEN; GOTO L2; CLOSE;
113     ST1.HD.TL.TL.HD.HD->REFPOINT;
114     WHICH((XEQCL<>COMPLEQCL),ST1.HD.TL.HD.VALOF)->XEQCL;
115     NIL->COMPLEQCL;
116     LISTDELETE(ST,ST1.HD)->LRRP.FNPROPS.TL;
```

```
117     ST1.HD.TL.TL.HD->ST2;
118     GOTO L3;
119
120   L2:
121     COMMENT '2. GET POTENTIAL REFPOINTS.'
122             '2.1 ANY EDGE WITH A SQUARE TAKEN IN IT.';
123     WHICH(£EDGES,LAMBDA; .VACANT.NOT; END)->ST1;
124
125     COMMENT '2.2 SQUARES TAKEN FIRST AND LAST.';
126     ((1.LASTWO)<>(0.LASTWO)<>ST1)->ST1;
127
128     COMMENT '2.3 FROM PREVREF, NON-PROSTRINGS FIRST.';
129     PREVREF->PT; NIL->ST;
130     LOOPIF PT.GIVEN THEN
131       PT.DEST->PT->ST2;
132       IF ST2=NIL THEN;
133       ELSEIF (ST2.HD.HD,ISA(LOCATION)) AND (BOARD,NOT(ISIN(ST2.HD)))
134         THEN
135         IF ("PROSTRING",ISIN(ST2.TL.TL.HD)) THEN
136           IF NOT("*",ISIN(ST1)) THEN
137             ("*"::(ST2.HD.HD::ST1))->ST1;
138           CLOSE;
139         ELSE ST2.HD.HD-->ST;
140         CLOSE;
141       CLOSE;
142     CLOSE;
143     IF NOT("*",ISIN(ST1)) THEN "*"-->ST1; CLOSE;
144     ((ST.REV)<>ST1)->ST2;
145
146   L3:
147     COMMENT '3. TRY TO RELATE TO ONE OF THE POINTS IN ST2.';
148     IF ST2=NIL THEN 0->OK; EXIT;
149     ST2.DEST->ST2->ST; COMPLEQCL->SAVE;
150     IF ST="*" THEN;
151       COMMENT 'JUST A MARKER, TO BE IGNORED.';
152       GOTO L3;
153     ELSEIF SPATIALRELATION(XEQCL,COMPLEQCL,ST) THEN
154       ->PPC.PPSELF; GOTO L4;
155     ELSE; GOTO L3;
156   CLOSE;
157
158   L4:
159     COMMENT '4. A RELATION HAS BEEN FOUND.'
160             '4.1 SEE IF THE REFPOINT IS RECENT ENOUGH TO BE IMPLICIT.';
161     IF ("*",ISIN(ST2)) THEN 1->OK;
162     ELSEIF (ST,ISA(LINES)) AND
163         ALL(1.LASTWO,ISIN(%£ST%)) OR (.MOVENO<3) THEN
164     1->OK;
165     ELSE 0->OK;
166     CLOSE;
167     IF OK THEN
168       [ADJG SIMPLE].COPYLIST->PPC.PPFE;
169       "MODIFIED"-->PPC.PPDAD.PPFE; MAKEADJG(PPC);
170     EXIT;
171
172     COMMENT '5.1 THE REFPOINT MUST BE MADE EXPLICIT.';
173     CONSNODE(NIL,NIL,[%ST%],PPC,NIL,NIL)->ST;
174     IF ("OPPOSITE",ISIN(PPC.PPSELF)) THEN
175       [NG OBJECT].COPYLIST->ST.PPFE;
176       MAKENG(ST)->OK;
```

```
177       ELSE [PREPG TO].COPYLIST->ST.PPFE;
178         MAKEPNG(ST)->OK;
179       CLOSE;
180       NIL->PPC.PPSON;
181       IF NOT(OK) THEN EXIT;
182       [ADJG COMPLEX].COPYLIST->PPC.PPFE;
183       ("QUALIFIED"::("ADJGQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
184       [%PPC.PPSELF.HD,ST%]->PPC.PPSELF;
185       MAKEADJG(PPC);
186     END;
187
```

```
FUNCTION LOCD6 REF PPC => OK;
COMMENT 'REFER TO THE MIDDLE OF THE BOARD.';

(REF.HD,EQUAL(SQUARE 5))->OK;
IF NOT(OK) THEN EXIT;
[%BOARD%]->PPC.PPSELF; [PREPG OF].COPYLIST->PPC.PPFE;
MAKEPNG(PPC)->OK; NIL->PPC.PPDAD.PPSON;
PPC.PPDAD.PPFE->REF;
LISTDELETE(REF,[NOUN PRONOUN PROREF PROSTRING])->REF;
("NOUN"::("QUALIFIED"::("PREPGQUAL"::REF)))->PPC.PPDAD.PPFE;
"MIDDLE"->NEWREF;
END;

FUNCTION LOCD7 REF PPC => OK;
COMMENT 'ATTEMPT REFERENCE TO THE LINE BY LAST TWO MOVES MADE.';
VARS PT ST ST1; 0->OK;
IF NOT(REF.HD,ISA(LINES)) THEN EXIT;
IF SOME(CANLIST.TL,LAMBDA; (.HD,ISA(LINES)); END) THEN EXIT;
1.LASTWO->ST;
IF ST.NULL THEN EXIT;
PREVREF->PT;
LOOPIF PT.GIVEN THEN PT.DEST->PT->ST1;
  IF ST1=NIL THEN;
  ELSEIF (ST1.HD.HD,ISIN(ST)) THEN;
  ELSEIF (ST1.HD.HD,ISA(LOCATION)) THEN 0->OK; RETURN;
  CLOSE;
CLOSE;

COMMENT 'GET THE LINE, IF ANY, WHICH THE LAST TWO MOVES ARE ON.';
APPLIST(£LINES,LAMBDA X;
  IF ST.ISLIST AND ALL(ST,LAMBDA Y; Y,ISIN(£X);END) THEN
    X->ST;
  CLOSE;
  END);
IF NOT(SAMEDATA(ST,ENTITY)) THEN EXIT;

COMMENT 'SEE IF THIS LINE IS THE REFERENT.';
IF EQUAL(REF.HD,ST) THEN [SAME]->PPC.PPSELF;

COMMENT 'SEE IF THE REFERENT CAN BE RELATED TO IT.';
ELSEIF SPATIALRELATION(XEQCL,COMPLEQCL,ST) THEN ->PPC.PPSELF;
ELSE RETURN;
CLOSE;

COMMENT 'SUCCESSS.';
[ADJG SIMPLE].COPYLIST->PPC.PPFE; "MODIFIED"-->PPC.PPDAD.PPFE;
MAKEADJG(PPC); NIL->PPC.PPDAD.PPSON;
1->OK;
END;



FUNCTION LOCD8 PPC => OK;
COMMENT 'IDENTIFY THE LINE BY SAYING WHEN THE MIDDLE OF IT'
        'WAS TAKEN';
VARS ST ST1; REF.HD->ST;
```

```
57    IF NOT(ST,ISA(LINES)) THEN 0->OK; EXIT;
58    £ST.TL.HD->ST;
59    IF ST.VACANT THEN 0->OK; EXIT;
60    £ST.TL.TL.HD->ST1;
61    IF ST1<3 OR ST1>(.MOVENO-3) THEN 1->OK;
62    ELSE 0->OK; RETURN;
63    CLOSE;
64
65    COMMENT 'MAKE THE RELATIVE CLAUSE.';
66    "CLAUSE"->PPC.PPUNIT;
67    [DEPENDENT FINITE RELATIVE].COPYLIST->PPC.PPFE;
68    IF CHOICEOF([0 1]) THEN "DANGLINGP";
69      IF CHOICEOF([0 1]) THEN
70        "EXPLIC"-->PPC.PPFE; "NOMREL"-->PPC.PPFE;
71      CLOSE;
72    ELSE "EXPLIC"; "PREPREL"-->PPC.PPFE;
73    CLOSE;
74    -->PPC.PPFE;
75    [%£ST.TL.HD,ST,"TAKE",[%ST%]%]->PPC.PPSELF;
76    IF ST1>(.MOVENO-3) THEN [PRADJ PRJUST PERFECTIVE].COPYLIST;
77    ELSE [APPADJ TIMEADJ TFIRST].COPYLIST;
78    CLOSE;
79    (<>PPC.PPFE)->PPC.PPFE;
80    (WHDEPTH-1)->WHDEPTH;
81    MAKES(PPC.PPSELF,PPC.PPFE,NIL,PPC)->OK;
82    (WHDEPTH+1)->WHDEPTH;
83    IF OK THEN
84      ("QUALIFIED"::("PREPGQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
85    CLOSE;
86    NIL->PPC.PPDAD.PPSON;
87    END;
88
89
90    FUNCTION LOCD9 REF PPC => OK;
91    COMMENT 'IF THE REF IS THE MIDDLE OF AN EDGE, FIRST TRIES TO`
92            'DESCRIBE IT AS SUCH. IF REF IS A CORNER, FIRST TRIES TO`
93            'MENTION THE EDGES TO WHICH IT IS COMMON. OTHERWISE`
94            'TRIES TO SAY WHAT REF LIES BETWEEN.';
95    VARS ST ST1 LOCDSW9; 0->LOCDSW9;
96    IF (REF.HD,ISA(LINES)) THEN 0->OK; EXIT;
97
98    IF ("DANGLINGP",ISIN(PPC.PPDAD.PPFE)) OR
99       ("PREPREL",ISIN(PPC.PPDAD.PPDAD.PPFE)) THEN;
100     COMMENT 'THE CURRENT REF IS A PART OF THE LINE REFERRED TO BY`
101             'THE NG NODE DOMINATING THE CURRENT ONE.';
102     COMMENT 'FETCH THE REFERENT.';
103     PTS(PTU,"NG",PPC.PPDAD)->OK;
104     IF NOT(OK) THEN 'ERROR 1 IN LOCD9`.PRSTRING; .POPREADY;
105     ELSE ->ST;
106     CLOSE;
107     WHICH(ST.PPSELF,
108       LAMBDA X; (REF.HD,ISIN(£X)); END)
109     ->REFPOINT;
110     IF REFPOINT.NULL THEN 'ERROR 2 IN LOCD9`.PRSTRING; .POPREADY; CLOSE;
111     SPATPART(XEQCL,REFPOINT.HD,PPC)->OK;
112     NIL->PPC.PPDAD.PPSON; RETURN;
113   CLOSE;
114
115   IF (REF.HD,ISA(CORNERS)) THEN; GOTO L1; CLOSE;
116   L0:
```

```
117       MAKERELNG(REF,XEQCL,PPC)->OK;
118       PPC.PPBRO->PPC.PPDAD.PPSON;
119       COMMENT 'IF SUCCESSFUL, FUNCTION SPATPART HAS AMENDED THE '
120               'FEATURES OF PPC.PPDAD ALREADY.';
121       1->LOCDSW9;
122       IF OK OR LOCDSW9 THEN EXIT;
123       1->LOCDSW9;
124
125     L1:
126       COMMENT 'FIND THE ENVIRONMENT OF REF.';
127       REF.HD->REFPOINT;
128     IF (REFPOINT,ISA(CORNERS)) THEN
129       WHICH(£EDGES,ADJACENT);
130     ELSE WHICH(£CORNERS,ADJACENT);
131     CLOSE;
132     ->ST;
133
134     COMMENT 'CHECK FOR CIRCULARITY.';
135     IF SOME(CANLIST, LAMBDA X; (INTERSECT(X,ST).NULL.NOT); END) THEN
136       0->OK; RETURN;
137     CLOSE;
138
139     COMMENT '3. MAKE FIRST ATTEMPT UPON IDENTIFYING THE CORNER.';
140     IF (REF.HD,ISA(CORNERS)).NOT THEN; GOTO L2; CLOSE;
141     CONSNODE(NIL,[PREPG TO].COPYLIST,ST,PPC,NIL,NIL)->ST1;
142     COMMENT 'FIX WHDEPTH TO PREVENT ABSURDITIES.';
143     (WHDEPTH+1)->WHDEPTH;
144     MAKEPNG(ST1)->OK; NIL->PPC.PPSON;
145     COMMENT 'UNFIX WHDEPTH.';
146     (WHDEPTH-1)->WHDEPTH;
147     IF OK THEN
148       [%"COMMON",ST1%]->PPC.PPSELF;
149       [ADJG COMPLEX].COPYLIST->PPC.PPFE;
150       MAKEADJG(PPC); NIL->PPC.PPDAD.PPSON;
151       ("QUALIFIED"::("ADJGQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
152       RETURN;
153     CLOSE;
154
155     COMMENT 'HAVING FAILED TO REFER TO THE CORNER, TRY NOW MENTIONING'
156             'THE SQUARES IT LIES BETWEEN.';
157     REF.HD->REFPOINT;
158     WHICH([%SQUARE 2,SQUARE 4,SQUARE 6,SQUARE 8%],ADJACENT)->ST;
159
160     L2:COMMENT 'IDENTIFY REF BY SAYING WHAT IT LIES BETWEEN.';
161     [PREPG BETWEEN].COPYLIST->PPC.PPFE; ST->PPC.PPSELF;
162     MAKEPNG(PPC)->OK; NIL->PPC.PPDAD.PPSON;
163     IF OK THEN
164       ("QUALIFIED"::("PREPGQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
165       RETURN;
166     CLOSE;
167
168     COMMENT 'IF REF IS A CORNER, TRY REFERRING TO IT AS "THE END OF...".';
169     IF LOCDSW9 OR (REF.HD,(ISA(CORNERS))).NOT THEN 0->OK; EXIT;
170     1->LOCDSW9; GOTO L0;
171     END;
172
```

```
[LOCD10]                          [ 20.48 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974


 1   FUNCTION LOCD10 XEQCL COMPLEQCL PPC => OK;
 2   COMMENT 'AS A METHOD OF LAST RESORT, REFERS TO THE REF. BY TAKING'
 3           'A CROSS BEARING ON IT FROM TWO REFERABLE POINTS.';
 4   VARS PPCST SAVE ST;
 5
 6   IF WHDEPTH>1 THEN 0->OK; EXIT;
 7   IF (REF.HD,ISA(SQUARES)) AND (REF.HD,ISA(CORNERS)).NOT THEN
 8     (WHDEPTH-1)->WHDEPTH;
 9     MAKERELNG(REF,XEQCL,PPC)->OK;
10     NIL->PPC.PPDAD.PPSON;
11     IF NOT(OK) THEN 'ERROR 0 IN LOCD10'.PRSTRING; .POPREADY; EXIT;
12     (WHDEPTH+1)->WHDEPTH;
13     RETURN;
14   CLOSE;
15
16   IF (REF.HD,ISA(LINES)) THEN REF;
17   ELSE WHICH(£EDGES,LAMBDA X; REF.HD,ISIN(£X); END);
18   CLOSE;
19   ->ST;
20   LRRP(ST)->OK;
21   IF NOT(OK) THEN 'ERROR 1 IN LOCD10'.PRSTRING; .POPREADY; EXIT;
22
23   IF (REF.HD,ISA(CORNERS)) THEN
24     (WHDEPTH-1)->WHDEPTH; LOCD9(REF,PPC)->OK; (WHDEPTH+1)->WHDEPTH;
25   EXIT;
26
27   COMMENT 'FNPROPS OF LRRP NOW CONTAINS DETAILS OF THE CROSS-BEARINGS.';
28   LRRP.FNPROPS.TL->SAVE;
29   SAVE.DEST.HD->SAVE->ST;
30   [ADJG COMPLEX]->PPC.PPFE; NIL->PPC.PPSELF;
31
32   IF (ST.TL.HD,ISIN(SAVE)) THEN;
33     COMMENT 'BOTH REFPOINTS HAVE A COMMON RELATION TO REF,'
34             'SO MAKE A COORDINATE PNG OR NG.';
35     CONSNODE(NIL,NIL,NIL,PPC,NIL,NIL)->PPCST;
36     IF ("ADJACENT",ISIN(ST)) THEN
37       [PREPG TO].COPYLIST->PPCST.PPFE;
38     ELSE [NG OBJECT].COPYLIST ->PPCST.PPFE;
39     CLOSE;
40     [%ST.TL.TL.HD.HD,SAVE.TL.TL.HD.HD%]->PPCST.PPSELF;
41     IF ("PREPG",ISIN(PPCST.PPFE)) THEN MAKEPNG(PPCST);
42     ELSE MAKENG(PPCST);
43     CLOSE;
44     ->OK; NIL->PPC.PPSON;
45     IF NOT(OK) THEN 'ERROR 2 IN LOCD10'.PRSTRING; .POPREADY; CLOSE;
46     [%ST.TL.HD,PPCST%]->PPC.PPSELF;
47     GOTO L2;
48   CLOSE;
49
50   COMMENT 'THE TWO REFPOINTS HAVE DIFFERENT RELATIONS TO REF,'
51           'SO MAKE A COMPLEX COORDINATE ADJG.';
52   "COORD"-->PPC.PPFE;
53   L1:
54   CONSNODE(NIL,NIL,NIL,PPC,NIL,NIL)->PPCST;
55   IF ("ADJACENT",ISIN(ST)) THEN [PREPG TO].COPYLIST;
56   ELSE [NG OBJECT].COPYLIST;
```

```
57    CLOSE;
58    ->PPCST.PPFE;
59    ST.TL.TL.HD->PPCST.PPSELF;
60    IF ("PREPG",ISIN(PPCST.PPFE)) THEN MAKEPNG(PPCST);
61    ELSE MAKENG(PPCST);
62    CLOSE;
63    ->OK; NIL->PPC.PPSON;
64    IF NOT(OK) THEN 'ERROR 3 IN LOCD10'.PRSTRING; .POPREADY; CLOSE;
65    [%ST.TL.HD,PPCST%]-->PPC.PPSELF;
66    IF SAVE.NULL THEN; PPC.PPSELF.REV->PPC.PPSELF; GOTO L2; CLOSE;
67    SAVE->ST; NIL->SAVE;
68    GOTO L1;
69
70    L2:
71    MAKEADJG(PPC); NIL->PPC.PPDAD.PPSON;
72    ("QUALIFIED"::("PREPGQUAL"::PPC.PPDAD.PPFE))->PPC.PPDAD.PPFE;
73    1->OK;
74    END;
75
```

```
[LOCFNS]                        [ 20.33 11 MAY 1974]
DTRACK 217    CREATED   20.30   11  5 1974

 1   FUNCTION RELATED Q1 FNSW Q2;
 2   COMMENT 'THE LOGIC OF THIS FUNCTION IS SET OUT ON THE DECISION'
 3           'CHART GOVERNING THE APPLICATION OF THE MODIFIERS'
 4           'ADJACENT AND OPPOSITE.';
 5
 6   IF (Q1,EQUAL(Q2)) THEN 0; EXIT;
 7   IF (Q1,ISA(DIAGONALS)) OR (Q2,ISA(DIAGONALS)) THEN 0; EXIT;
 8   IF (Q2,EQUAL(R2)) OR (Q2,EQUAL(R2)) THEN 0; EXIT;
 9   IF (Q1,ISA(SQUARES)) AND (Q2,ISA(EDGES)) THEN 0; EXIT;
10
11   IF (Q1,ISA(EDGES)) THEN
12    IF (Q2,ISA(EDGES)) THEN
13     IF FNSW THEN SOME(£Q1,LAMBDA;ISIN(£Q2);END);
14     ELSE NO(£Q1,LAMBDA; ISIN(£Q2);END); CLOSE;
15     RETURN;
16    ELSEIF (Q2,ISA(CORNERS)) THEN
17     IF (Q2,ISIN(£Q1)) THEN FNSW;
18     ELSE NOT(FNSW); CLOSE;
19     RETURN;
20    ELSEIF FNSW THEN 0; RETURN;
21    ELSE (£Q1.TL.HD)->Q1; GOTO L1;
22    CLOSE;
23   ELSEIF (Q1,ISA(LINES)) THEN
24    IF NOT(FNSW) THEN 0; EXIT;
25    IF (Q2,ISA(SQUARES)) THEN 0; EXIT;
26    IF NO(£Q1,LAMBDA;ISIN(£Q2);END) THEN
27     (£Q1.TL.HD)->Q1;
28     (£Q2.TL.HD)->Q2;
29     GOTO L1;
30    ELSE 0; RETURN;
31    CLOSE
32   CLOSE;
33
34   L1:
35    COMMENT 'Q1 IS, OR IS TO BE TREATED AS, A SQUARE.';
36    VARS I1 I2 J1 J2;
37    £Q1.HD.COORDS->J1->I1; £Q2.HD.COORDS->J2->I2;
38    (I1-I2)↑2->I1; (J1-J2)↑2->J1;
39    IF (Q1,ISA(CORNERS)) THEN
40     IF (Q2,ISA(CORNERS)) THEN
41      IF FNSW THEN (I1+J1)=4.0;
42      ELSE (I1+J1)=8.0;
43      CLOSE; RETURN;
44     ELSEIF FNSW THEN BOOLOR(((I1+J1)=1.0),((I1+J1)=2.0));
45     ELSE 0;
46     CLOSE;
47     RETURN;
48    ELSEIF FNSW THEN BOOLOR(((I1+J1)=1.0),((I1+J1)=2.0));
49    ELSE (I1+J1)=4.0;
50    CLOSE;
51   END;
52
53   FUNCTION ADJACENT Q1; RELATED(Q1,TRUE,REFPOINT); END;
54
55   FUNCTION OPPOSITE Q1; RELATED(Q1,FALSE,REFPOINT); END;
56
```

```
57
58   FUNCTION MIDDLE Q1;
59   VARS REFX; £REFPOINT->REFX;
60   L: IF LENGTH(REFX)<3 THEN 0; EXIT;
61      REFX.TL.REV.TL->REFX;
62      IF REFX.LENGTH=1 THEN EQUAL(Q1,HD(REFX)); EXIT;
63      GOTO L;
64   END;
65
66   FUNCTION END1 Q1;
67   VARS REFX; £REFPOINT->REFX;
68   IF LENGTH(REFX)=3 THEN
69    BOOLOR(EQUAL(Q1,HD(REFX)),EQUAL(Q1,HD(REV(REFX))));
70   ELSE FALSE; CLOSE;
71   END;
72
```

```
[LRRP]                    [ 20.48 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974


    1   FUNCTION LRRP REF => OK;
    2   VARS LIST ST ST1 ST2 SW OJT;
    3   NIL->OUT; 0->SW;
    4   (1.LASTWO<>0.LASTWO).CONDENSE->ST;
    5   LISTDELETE(ST,SQUARE 5)->ST;
    6   IF ST.LENGTH<2 THEN 0->OK; EXIT;
    7   REF.LENGTH->N;
    8
    9   COMMENT 'GET A USEFUL PAIR.';
   10   L0:
   11     IF ST.NULL THEN; GOTO L1; CLOSE;
   12     ST.DEST->ST->ST1;
   13     WHICH(ST,LAMBDA X; ((£X.HD + £ST1.HD)/=10); END)->LIST;
   14     IF LIST.NULL AND ST.LENGTH=1 THEN
   15       'ERROR 1 IN LRRP'.PRSTRING; .POPREADY;
   16     ELSEIF LIST.GIVEN THEN ((ST1::LIST)<>OUT)->OUT;
   17     CLOSE;
   18   GOTO L0;
   19
   20   L1:
   21     OUT.CONDENSE->ST;
   22     IF ST.LENGTH<2 THEN 'ERROR 2 IN LRRP'.PRSTRING; .POPREADY; EXIT;
   23     COMMENT 'ST NOW CONTAINS AT LEAST TWO USEFUL REFERENCE POINTS.';
   24     NIL->OUT;
   25     COMMENT 'TRY TO GET A BEARING FROM THE MIDDLE OF AN EDGE.';
   26     WHICH(ST,
   27       LAMBDA REFPOINT;
   28         IF (REFPOINT,ISA(CORNERS)) THEN 0; EXIT;
   29         IF (REFPOINT,ISIN(REF.HD.CONTENTS)) THEN 1->SW; 1;          IF REF;
ENGTH=2 THEN REF.TL->REF; CLOSE;
   30         EXIT;
   31         IF NONMAP([%REF.HD%],NIL,[%OPPOSITE%]) THEN REF.HD->ST2;
   32         ELSEIF N=2 AND NONMAP([%REF.TL.HD%],NIL,[%OPPOSITE%]) THEN
   33           .ERASE; REF.TL.HD->ST2;
   34         ELSE 0; RETURN;
   35         CLOSE;
   36         IF N=2 THEN LISTDELETE(REF,ST2)->REF; CLOSE;
   37         [%[%ST2%],"OPPOSITE",[%REFPOINT%]%]-->OUT;
   38         1;
   39       END);
   40     ->ST2; LISTDELETE(ST,ST2)->ST;
   41
   42     COMMENT 'NOW MENTION ADJACENCY.';
   43     IF ST.NULL AND (OUT.LENGTH+SW)=2 THEN; GOTO L3; CLOSE;
   44     LOOPIF ST.GIVEN THEN ST.DEST->ST->REFPOINT;
   45       L2:
   46       IF NONMAP([%REF.HD%],NIL,[%ADJACENT,OPPOSITE%]) THEN ->ST2;
   47         [%[%REF.HD%],ST2.HD,[%REFPOINT%]%]-->OUT;
   48       ELSEIF (REFPOINT,ISA(SQUARES)) THEN
   49         HD(WHICH(£EDGES,LAMBDA X;REFPOINT,ISIN(£X);END))->REFPOINT;
   50         GOTO L2;
   51       CLOSE;
   52       IF OUT.LENGTH+SW = 2 THEN; GOTO L3;
   53       ELSEIF N=2 AND REF.LENGTH=2 THEN REF.TL->REF;
   54       CLOSE;
   55     CLOSE;
```

```
56    IF (REF.HD,EQUAL(REFPOINT)) THEN; GOTO L3; CLOSE;
57    'ERROR 3 IN LRRP'.PRSTRING; .POPREADY;
58
59    L3:
60    COMMENT 'ORDER OUT.';
61      IF N=1 AND (OUT.HD.TL.TL.HD.HD,ISA(EDGES)).NOT THEN
62        REV(OUT)->OUT;
63      CLOSE;
64    OUT->LRRP.FNPROPS.TL; 1->OK;
65    END;
66
```

```
[MACROS]                          [ 20.33 11 MAY 1974]
DTRACK 217     CREATED    20.30    11  5 1974


  1    MACRO ==>; [.MACRESULTS;].MACRESULTS; END;
  2
  3    MACRO S; [.].MACRESULTS; END;
  4
  5    MACRO ISIN; [MEMBER].MACRESULTS; END;
  6
  7    MACRO -->;
  8    VARS X; READTO(";")->X;
  9    ([::]<>X<>[->]<>X<>[;])==>;
 10    END;
 11    FUNCTION BRACKETREAD LB RB => OUT;
 12    VARS X I; NIL->OUT; 0->I;
 13    L: .ITEMREAD->X;
 14     IF (X,ISIN(LB)) THEN I+1;
 15     ELSEIF (X,ISIN(RB)) THEN I-1;
 16     ELSE I; CLOSE; ->I;
 17     X-->OUT;
 18     IF I>0 THEN; GOTO L; CLOSE;
 19    OUT.REV->OUT;
 20    END;
 21
 22    MACRO £;
 23    VARS X;
 24     .ITEMREAD->X;
 25    [%"CONTENTS","(",X%]==>;
 26    IF X.VALOF.ISFUNC THEN BRACKETREAD([( (%],[) %)])==>;CLOSE;
 27    [)]==>;
 28    END;
 29
 30    MACRO :+;
 31    VARS X XOUT FIRST;
 32    NIL->XOUT; 1->FIRST;
 33    L0: .ITEMREAD-->XOUT;
 34     IF XOUT.HD="NONOP" THEN; GOTO L0; CLOSE;
 35    L1: BRACKETREAD([( (%],[) %)])->X;
 36     IF LENGTH(X)>1 THEN (X.REV<>XOUT)->XOUT;
 37     ELSE X.HD-->PROGLIST; CLOSE;
 38     IF FIRST=1 THEN; ","-->XOUT; 0->FIRST; GOTO L0; CLOSE;
 39    ([FNCOMP(]<>(XOUT.REV)<>[)])==>;
 40    END:
 41
 42
 43    MACRO OF;
 44    VARS X L1 L2 FIRST;
 45    [%"(","(%","[","[%"%]->L1;  [%")","%)","]","%]"%]->L2;
 46    1->FIRST; [(]==>;
 47    L: .ITEMREAD->X;
 48     IF (X,ISIN(L1)) THEN X-->PROGLIST;
 49      BRACKETREAD(L1,L2)<>[)]; ==>; EXIT;
 50     IF FIRST=1 THEN [%X%]==>; 0->FIRST; GOTO L; CLOSE;
 51    [%")",X%]==>;
 52    END;
 53
```

```
[MAKEADJG]                          [ 20.41 11 MAY 1974]
DTRACK 217     CREATED    20.38    11  5 1974


 1   FUNCTION MAKEADJG PPC;
 2   VARS ST;
 3   "ADJG"->PPC.PPUNIT;
 4
 5   IF ("COORD",ISIN(PPC.PPFE)) THEN
 6     ADDTO(CONSNODE(NIL,LISTDELETE(PPC.PPFE,"COORD"),
 7                    PPC.PPSELF.HD,PPC,NIL,NIL));
 8     MAKEADJG(PPC.PPSON);
 9     ADDTO(CONSNODE("WORD",NIL,[AND],PPC,NIL,NIL));
10     ADDTO(CONSNODE(NIL,LISTDELETE(PPC,PPFE,"COORD"),
11                    PPC.PPSELF.TL.HD,PPC,NIL,NIL));
12     MAKEADJG(PPC.PPSON);
13   EXIT;
14
15   ADDTO(CONSNODE("WORD",NIL,(PPC.PPSELF.HD::NIL),PPC,NIL,NIL));
16   IF ("COMPLEX",ISIN(PPC.PPFE)) THEN
17     PPC.PPSELF.TL.HD->ST; PPC->ST.PPDAD;
18     ADDTO(ST);
19   CLOSE;
20   END;
21
```

```
[MAKENG]                    [ 20.49 11 MAY 1974]
DTRACK 217    CREATED   20.46   11  5 1974

 1   FUNCTION MAKENG PPC => OK;
 2   VARS COMPLEQCL FLIST FUNSET NEWREF PT REF SAVESET ST X XEQCL;
 3
 4   COMMENT 'IF THE NG IS A RELATIVE GOVERNED BY A DANGLING PREP'
 5           'IT MAY BE NECESSARY TO GET THE REFERENT FROM CANLIST.';
 6   IF PPC.PPSELF.NULL AND ("RELATIVE",ISIN(PPC.PPFE)) THEN
 7     CANLIST.HD->PPC.PPSELF;
 8   CLOSE;
 9
10   PPC.PPFE->FLIST; PPC.PPSELF->REF;
11   "NG"->PPC.PPUNIT; 0->OK;NIL->XEQCL; NIL->SAVESET;
12
13   COMMENT '1. SEE WHETHER NG IS A DEPENDENT CLAUSE.';
14   PPC.PPSELF->ST;
15   IF ST.LENGTH=2 AND ST.TL.HD.ISLIST AND ("DEPENDENT",ISIN(ST.TL.HD))
16     THEN
17     CONSNODE("S",NIL,NIL,PPC,NIL,NIL)->ST;
18     MAKES(PPC.PPSELF.HD,
19      ([USTAND USTANDI USTANDA]<>PPC.PPSELF.TL.HD),NIL,ST);
20     "CLAUSE"-->FLIST; ST-->SAVESET; NIL->ST;
21
22     COMMENT 'SEE WHETHER A DETERMINER IS NEEDED.';
23     IF ("USTAND",ISIN(PPC.PPSELF.TL.HD)) THEN "NOTDET"-->FLIST;
24     ELSE ("DETERMINED"::("DEFINITE"::("POSSESS"::FLIST)))->PPC.PPFE;
25       CONSNODE(NIL,NIL,NIL,PPC,NIL,NIL)->ST;
26       [%PPC.PPSELF.HD.HD%]->REF; NIL->FLIST; REF.PERSONQ;
27       IF PRONOUN(REF) THEN
28         ([ARTICLE POSSESS SG].COPYLIST)->FLIST; REF.PERSONQ;
29         REF.ANIMATEQ; FLIST->ST.PPFE; MAKEWORD(ST);
30       ELSE "DET"->ST.PPUNIT;
31         ADDTO(CONSNODE(NIL,[NG GENITIVE].COPYLIST,REF,ST,NIL,NIL));
32        MAKENG(ST.PPSON); .ERASE;
33       CLOSE;
34       PPC.PPFE->FLIST;
35     CLOSE;
36     1->OK;
37     IF NOT(ST=NIL) THEN ST-->SAVESET; CLOSE;
38     GOTO L3;
39   CLOSE;
40
41     COMMENT '2. SEE IF THE NG IS COORDINATE.';
42     EQCL(REF)->XEQCL; 0->COMPLEQCL;
43     IF REF.LENGTH>1 AND SOME(REF,LAMBDA;NOT(ISIN(XEQCL));END) THEN
44       [%"COORD"%]->PPC.PPFE;
45       ADDTO(CONSNODE("NG",FLIST,[%REF.HD%],PPC,NIL,NIL)):
46       PREVREF.COPYLIST->SAVESET;
47       IF MAKENG(PPC.PPSON) THEN
48         ADDTO(CONSNODE("WORD",[WORD LINK SEQ],[AND],PPC,NIL,NIL));
49         ADDTO(CONSNODE("NG",FLIST,REF.TL,PPC,NIL,NIL));
50         MAKENG(PPC.PPSON)->OK;
51       ELSE PPC.PPBRO->PPC.PPDAD.PPSON;
52       CLOSE;
53       IF NOT(OK) THEN SAVESET->PREVREF; CLOSE;
54     EXIT;
55
56     COMMENT '3. TRAVERSE PART OF THE SYSTEMS NETWORK.';
```

```
57      "NOMINAL"-->FLIST; REF.SG; REF.PERSONQ; REF.ANIMATEQ;
58      IF PRONOUN(REF) THEN
59        REF.HD.NNAME->NEWREF; "PRONOUN"-->FLIST;
60        IF NEWREF=NIL THEN REF.HD.TYPE.HD.NNAME->NEWREF; CLOSE;
61        1->OK; GOTO L3;
62      ELSEIF PROPERNOUN(REF) THEN
63        REF.HD.NNAME->NEWREF; [%NEWREF%]->PPC.PPSELF;
64        "PROPERNOUN"-->FLIST; 1->OK; GOTO L3;
65      CLOSE;
66
67      COMMENT '4. THE HEAD IS A COMMON NOUN, PROSTRING, ADJ OR NUMBER.';
68      REF.HD.TYPE.HD->X; NNAME(X)->NEWREF;
69
70      COMMENT '5. SEE IF THE HEAD IS NUMBER, NOUN, OR PRONOUN.';
71      IF NUMDET(REF,XEQCL) THEN;
72      ELSEIF PROSTRING(REF) THEN
73        ("PRONOUN"::("PROSTRING"::FLIST))->FLIST;
74      ELSE ("NOUN"::FLIST)->FLIST;
75      CLOSE;
76      IF (XEQCL,COINCIDE(£X)) THEN; 1->OK; GOTO L3; CLOSE;
77
78      COMMENT '6. A MODIFIER OR QUALIFIER OR NUMBER IS NEEDED.';
79      FLIST->PPC.PPFE; XEQCL->PPC.PPSELF;
80      IF ("NUMBERED",ISIN(FLIST)) AND NOT(REF,COINCIDE(XEQCL)) THEN
81        GOTO L2;
82      CLOSE;
83      IF WHDEPTH=3 THEN 0->OK; PPC.PPBRO->PPC.PPDAD.PPSON; EXIT;
84      CONSNODE("UNDEF",FLIST,[%XEQCL,X,REF%],PPC,NIL,NIL)->ST;
85      CANLIST.LENGTH->PT;
86      LOCDESCR(ST,WHDEPTH+1)->OK;
87      IF CANLIST.GIVEN AND CANLIST.LENGTH>PT THEN
88        CANLIST.TL->CANLIST;
89      CLOSE;
90      IF NOT(OK) THEN PPC.PPBRO->PPC.PPDAD.PPSON; EXIT;
91      PPC.PPSELF->XEQCL; PPC.PPFE->FLIST; ST-->SAVESET;
92      GOTO L3;
93
94
95      COMMENT '8. THE REST HAVING FAILED, TRY A PREPG QUALIFIER.';
96    L2:
97      CONSNODE("PREPG",[PREPG OF].COPYLIST,XEQCL,PPC,NIL,NIL)->ST;
98      MAKEPNG(ST)->OK; NIL->PPC.PPSON;
99      IF OK THEN
100       ("QUALIFIED"::("PREPGQUAL"::FLIST))->FLIST; ST-->SAVESET;
101     CLOSE;
102
103   L3: COMMENT 'IF NOT OK, ABANDON THIS NG.';
104     IF NOT(OK) THEN PPC.PPBRO->PPC.PPDAD.PPSON; EXIT;
105
106     COMMENT '9. SEE IF XEQCL HAS BEEN ALTERED BY LOCDESCR.';
107     IF (REF,COINCIDE(XEQCL)) THEN
108       LISTDELETE(FLIST,"NUMBERED")->FLIST;
109     CLOSE;
110
111     COMMENT '10. TRAVERSE THE DEF/INDEF SYSTEM.';
112     DEFQ(REF);
113     COMMENT '11. REALISE CONSTITUENT STRUCTURE BY FTR & SB RULES.';
114     FLIST.COPYLIST->PPC.PPFE;
115     REALISE(FLIST,NGFTRRS)->FUNSET;
116     SB1(FUNSET,NGSBRS1)->FUNSET;
```

```
117     SB2(FUNSET,NGSBRS2)->FUNSET;
118   IF TESTSW4 THEN
119     2.NL;
120     'REF:    '.PRSTRING;
121       IF ("CLAUSE",ISIN(PPC.PPFE)) THEN "CLAUSE".PR;
122       ELSE REF.HD.NNAME.PR; 3.SP; £(REF.HD).PR;
123       CLOSE; 1.NL;
124     'FTS:    '.PRSTRING; PPC.PPFE.PR; 1.NL;
125     'FNS:    '.PRSTRING; FUNSET.PR; 3.NL;
126   CLOSE;
127     FUNSET->PT;
128     LOOPIF PT.GIVEN THEN
129       IF PT.HD.ISLIST.NOT THEN (PT.HD::NIL)->PT.HD; CLOSE;
130       REALISE(PT.HD,NGFURRS)->FLIST;
131       IF ("WORD",ISIN(FLIST)) AND ("POSSESS",NOT(ISIN(FLIST))) THEN
132         ADDTO(CONSNODE(NIL,FLIST,NIL,PPC,NIL,NIL));
133         MAKEWORD(PPC.PPSON);
134       ELSEIF INTERSECT([ADJG PREPG CLAUSE POSSESS],FLIST).GIVEN THEN
135         SAVESET.DEST->SAVESET->ST; NIL->ST.PPBRO; ADDTO(ST);
136         FLIST<>PPC.PPSON.PPFE->PPC.PPSON.PPFE;
137       CLOSE;
138     PT.TL->PT;
139     CLOSE;
140
141
142
143   COMMENT 'AMEND "A" BEFORE "OTHER" OR A VOWEL.';
144   IF PPC.PTLT THEN ->ST;
145     IF SAMEDATA(ST,ROOT) AND ST.PPSELF.HD="A" THEN
146       ST.PTYBRO.ERASE; ->X; X.PTLT.ERASE; ->PT;
147       IF PT.PPSELF.HD="OTHER" THEN
148         "ANOTHER"->PT.PPSELF.HD; NIL->X.PPBRO;
149       ELSEIF [%PT.PPSELF.HD.DESTWORD%].HD.VOWEL THEN
150         "AN"->ST.PPSELF.HD;
151       CLOSE;
152     CLOSE;
153   CLOSE;
154   IF NOT("CLAUSE",ISIN(PPC.PPFE)) THEN
155     [%REF,NEWREF,PPC.PPFE%]-->PREVREF;
156   CLOSE;
157   UPDATE1(PREVREF);
158   END;
159
```

```
[MAKEPNG]                          [ 20.4 11 MAY 1974]
DTRACK 217     CREATED    20.38    11   5 1974


    1   FUNCTION MAKEPNG PPC => OK;
    2   VARS ST; "PNG"->PPC.PPUNIT;
    3   COMMENT '1. CHECK FOR A PREPG INTERVENING IN THE SURFACE STRING'
    4            'BETWEEN THE HEAD OF THE DOMINATING NG AND THIS PREPG.';
    5   PPC.PPBRO->PPC.PPDAD.PPSON;
    6   NOT(PTS(PTDL,"PREPG",PPC.PPDAD))->OK;
    7   IF NOT(OK) THEN .ERASE; EXIT;
    8   PPC->PPC.PPDAD.PPSON;
    9
   10   COMMENT '2. IF THE PREPG IS A RELATIVE PRONOUN, GET THE '
   11            'ANTECEDENT FROM THE HEAD OF CANLIST.';
   12   IF ("RELATIVE",ISIN(PPC.PPFE)) THEN
   13     CANLIST.HD->PPC.PPSELF;
   14   CLOSE;
   15
   16   COMMENT '3. REALISE FEATURES.';
   17   REALISE(PPC.PPFE,PREPGFTR)->ST;
   18
   19   COMMENT '4. BUILD STRUCTURE.';
   20   SB2(ST,PREPGSB)->ST;
   21   ADDTO(CONSNODE("WORD",REALISE(ST.HD,PREPGFUR),NIL,PPC,NIL,NIL));
   22   MAKEWORD(PPC.PPSON);
   23   ST.TL->ST;
   24   IF ST.NULL THEN; 1->OK; EXIT;
   25   ADDTO(CONSNODE("NG",REALISE(ST,PREPGFUR),PPC.PPSELF,PPC,NIL,NIL));
   26   MAKENG(PPC.PPSON)->OK;
   27
   28   COMMENT 'IF A FAILURE, PRUNE TREE.';
   29   IF NOT(OK) THEN PPC.PPBRO->PPC.PPDAD.PPSON; CLOSE;
   30   END;
   31
```

```
[MAKERELNG]                          [ 20.35 11 MAY 1974]
DTRACK 217     CREATED    20.31    11  5 1974


  1    FUNCTION MAKERELNG REF XEQCL PPC;
  2    VARS ST ST1;
  3    IF (REF.HD,NOT(ISA(SQUARES))) THEN 0; EXIT;
  4    WHICH(£EDGES,LAMBDA X; REF.HD,ISIN(£X); END)->ST;
  5    IF ST.NULL THEN [%BOARD%]->ST; CLOSE;
  6    LOOPIF ST.GIVEN THEN ST.DEST->ST->ST1;
  7     NIL->PPC.PPSELF; NIL->PPC.PPSON;
  8     IF SPATPART(XEQCL,ST1,PPC) THEN 1; EXIT;
  9    CLOSE;
 10    0;
 11    END;
 12
```

```
[MAKES]                    [ 20.42 11 MAY 1974]
DTRACK 217    CREATED   20.39   11  5 1974


 1   FUNCTION MAKES DATA FEATURES ADJUNCT PPC => OK;
 2   COMMENT 'THIS FN DEFINES FUNCTION(S) TO MAKE THE CLAUSE(S)
 3            EXPRESSING THE DATA, ADDING THE ADJUNCT TO THE`
 4           'FIRST MAJOR CLAUSE, AND THEN MAKES THE CLAUSE(S).`;
 5   VARS ST ST1 ST2 FUNSET PPCD;
 6   FEATURES.COPYLIST->FEATURES; 0->OK;
 7
 8   COMMENT '1. DETERMINE WHETHER THE DATA NEEDS A CLAUSE ENVIRON.`;
 9   IF DATA.HD.ISLIST AND DATA.HD.NULL.NOT THEN
10     DATA.DEST->DATA->ST;
11     [ DEPENDENT FINITE PAST BOUND THOBOUND]->ST1;
12     IF ST.HD="IF" THEN
13       (ST1<>[MODAL MODALPOS PERFECTIV])->ST1;
14       ("ENVIRADJ"::("THOADJ"::FEATURES))->FEATURES;
15     ELSEIF ST.HD="ALTHOUGH" THEN
16       ("ENVIRADJ"::("THOADJ"::FEATURES))->FEATURES;
17     CLOSE;
18     [%ST.TL,ST1%]-->ADJUNCT;
19   CLOSE;
20
21   COMMENT '2. COORDINATE STRUCTURE IS REQUIRED IF`
22           ' A.  ONLY ONE MOVE IN WHOLE SENTENCE, OR`
23           ' B.  TWO EQUIVALENT ASPECTS.`;
24   IF DATA.LENGTH=4 THEN; GOTO L0; CLOSE;
25   IF ROOT.PPSELF.LENGTH=1 OR DATA.TL.TL.TL.TL.HD.PROVAL>1 THEN;
26   ELSE; GOTO L0;
27   CLOSE;
28   IF DATA.TL.TL.TL.TL.HD.PROVAL>1 THEN;
29     COMMENT 'REVERSE ORDER OF THE VERBS.`;
30     DATA.TL.TL.TL.TL->ST; NIL->DATA.TL.TL.TL.TL;
31     DATA.TL.TL->ST.TL.TL; ST->DATA.TL.TL;
32   CLOSE;
33     [COORD LINKED CONJUNC BINARY]->PPC.PPFE;
34     ADDTO(CONSNODE("S",NIL,NIL,PPC,NIL,NIL));
35     (DATA.HD::(DATA.TL.HD::DATA.TL.TL.TL.TL))->ST;
36     MAKES(ST,FEATURES,ADJUNCT,PPC.PPSON)->OK;
37     IF NOT(OK) THEN EXIT;;
38     ADDTO(CONSNODE("WORD",NIL,[AND],PPC,NIL,NIL));
39     ADDTO(CONSNODE("S",NIL,NIL,PPC,NIL,NIL));
40     DATA.REV.TL.TL.REV->DATA;
41     MAKES(DATA,[ USTAND USTANDI USTANDA],NIL,PPC.PPSON)->OK;
42     IF NOT(OK) THEN NIL->PPC.PPSON;CLOSE;
43   RETURN;
44
45   L0:
46   COMMENT 'EVIDENTLY A SIMPLE CLAUSE`;
47   ("SIMPLE"::("CLAUSE"::FEATURES))->FEATURES;
48
49   COMMENT '4. TRAVERSE [TRANSITIVITY] SYSTEMS IF NECESSARY.`;
50   IF ("EXT",ISIN(FEATURES)) THEN;
51   ELSEIF VERBTYPE(DATA.TL.TL.HD)=1 THEN
52     ("EXT"::("DESCR"::FEATURES))->FEATURES;
53     IF CHOICEOF([0 1]) THEN "OP"; ELSE "MID"; CLOSE;
54     -->FEATURES;
55   ELSE ("EXT"::("EFF"::FEATURES))->FEATURES;
56     COMMENT 'SELECT RECEPTIVE IF ACTOR/INITIATOR ABSENT.`;
```

```
57      IF DATA.HD=NIL THEN
58        ("USTANDA"::("RECEPTIV"::FEATURES))->FEATURES;
59      ELSE "OP"-->FEATURES;
60        IF (DATA.TL.TL.HD,ISIN([WIN DRAW])) AND CHOICEOF([0 1]) THEN
61           "GINTRANS";
62        ELSE "GTRANS";
63        CLOSE;
64        -->FEATURES;
65      CLOSE;
66    CLOSE;
67
68    COMMENT '5. NOW SEE WHETHER APPENDIX ADJUNCT`;
69    IF DATA.LENGTH>4 THEN "APPADJ"-->FEATURES;
70      COMMENT 'SELECT METHOD OR ACCOMP ACC. WHETHER MAIN CLAUSE &`
71              'ADJUNCT SHARE SUBJECT.`;
72      IF ("DESCR",ISIN(FEATURES)) AND ("MID",ISIN(FEATURES)) THEN
73        NIL->ST; "ACCOMPADJ";
74      ELSE [USTAND USTANDI USTANDA]->ST; "METHODADJ";
75      CLOSE;
76      -->FEATURES;
77      [DEPENDENT NONFIN ING PARTICIPLE]<>ST->ST;
78      [%(DATA.HD::(DATA.TL.HD::DATA.TL.TL.TL.TL)),ST%]-->ADJUNCT;
79      NIL->DATA.TL.TL.TL.TL;
80    CLOSE;
81
82    COMMENT '6. TRAVERSE [MOOD] SYSTEMS IF NECESSARY.`;
83    IF INTERSECT([PRESENT NONFIN],FEATURES).NULL THEN
84      "PAST"-->FEATURES;
85    CLOSE;
86    IF ("DEPENDENT",ISIN(FEATURES)) OR ("INDEP",ISIN(FEATURES)) THEN;
87    ELSE ([INDEP INDIC DECLAR]<>FEATURES)->FEATURES;
88    CLOSE;
89
90    COMMENT 'STORE COMPLETED FEATURE SET`;
91    FEATURES.COPYLIST->PPC.PPFE;
92    COMMENT '7. REALISE FEATURES IN FUNCTION SET.`;
93    REALISE(FEATURES,CLFTRRS)->FUNSET;
94    COMMENT '8. ORDER FUNCTION SET TO FUNCTION BUNDLE LIST.`;
95    SB1(FUNSET,CLSBRS1)->FUNSET;
96    SB2(FUNSET,CLSBRS2)->FUNSET;
97    IF TESTSW3 THEN
98    2.NL;'DATA:     `.PRSTRING; DATA.PR; 1.NL;
99         'FTS:      `.PRSTRING; PPC.PPFE.PR; 1.NL;
100        'ADJUNCT   `.PRSTRING; ADJUNCT.PR; 1.NL;
101        'FUNSET    `.PRSTRING; FUNSET.PR; 4.NL;
102   CLOSE;
103
104   COMMENT '9 CONSTRUCT IMMEDIATE CONSTITUENTS OF THE CLAUSE.`
105           'FIRST GET DETAILS OF SUBJECT FOR FINITE VERB.`;
106   .MAKESFN1;
107
108     COMMENT '10. NOW MAKE EACH CONSTITUENT.`;
109   FUNSET->ST2; 1->OK;
110   LOOPIF ST2.GIVEN THEN;
111     ADDTO(CONSNODE(NIL,NIL,NIL,PPC,NIL,NIL)); PPC.PPSON->PPCD;
112     COMMENT '11. REALISE FUNCTIONS IN FEATURES.`;
113     IF ST2.HD.ISLIST.NOT THEN ST2.HD::NIL->ST2.HD; CLOSE;
114     REALISE(ST2.HD.COPYLIST,CLFURRS)->PPCD.PPFE;
115     COMMENT '12. STORE SEMANTIC DATA, IF ANY, FOR THIS NODE.`;
116     APPLIST(ST2.HD,
```

```
117       LAMBDA F;
118         IF PPCD.PPSELF.NULL AND
119           F.IDENTPROPS=0 AND F.VALOF.ISFUNC THEN
120           APPLY(F.VALOF);
121           ->PPCD.PPSELF;
122           IF (F="ENVIR") THEN
123             (PPCD.PPSELF.TL.HD<>PPCD.PPFE)->PPCD.PPFE;
124             PPCD.PPSELF.HD->PPCD.PPSELF;
125           CLOSE;
126         CLOSE;
127       END);
128     COMMENT '13.  NOW MAKE THE CONSTITUENT.';
129     PPCD.PPFE->ST;
130     WHICH([CLAUSE NG PREPG VG WORD],LAMBDA X; X,ISIN(ST); END).HD
131     ->PPCD.PPUNIT;
132     IF PPCD.PPUNIT="CLAUSE" THEN
133       MAKES(PPCD.PPSELF,ST,NIL,PPCD)->OK;
134       IF EQUAL(PPCD.PPSELF.TL.HD,MOVES.HD.TL.HD) THEN
135         NIMMT(MOVES.HD.HD,MOVES.HD.TL.HD); MOVES.TL->MOVES;
136       CLOSE;
137     ELSEIF PPCD.PPUNIT="NG" THEN
138       IF ("OBJECT",ISIN(PPCD.PPFE)) AND (DATA.TL.TL.HD="TAKE") THEN
139         TAKEFN->ENVIRON;
140       CLOSE;
141       MAKENG(PPCD)->OK;
142       IDENTFN->ENVIRON;
143     ELSEIF PPCD.PPUNIT="PREPG" THEN
144       MAKEPNG(PPCD)->OK;
145     ELSEIF PPCD.PPUNIT="VG" THEN
146       MAKEVG(PPCD);
147     ELSEIF PPCD.PPUNIT="WORD" THEN
148       MAKEWORD(PPCD);
149     CLOSE;
150     ST2.TL->ST2;
151     IF NOT(OK) THEN NIL->PPC.PPSON; EXIT;
152   CLOSE;
153   END;
154
155
```

```
[MAKESFN1]                          [ 20.48 11 MAY 1974]
DTRACK 217    CREATED   20.45   11  5 1974


 1   FUNCTION MAKESFN1;
 2   VARS FLIST ST PT; NIL->FLIST;
 3   IF BFIND(FUNSET,"FINITE") THEN
 4     ->PT;
 5     IF PT.HD.ISLIST.NOT THEN [%PT.HD%]->PT.HD; CLOSE;
 6   ELSE RETURN;
 7   CLOSE;
 8
 9   IF BFIND(FUNSET,"SUBJECT") THEN .HD->ST;
10     WHICH([ACTOR INITR GOAL],LAMBDA; ISIN(ST); END).HD.VALOF.APPLY;
11     ->ST;
12     IF ST.LENGTH=1 THEN "SINGULAR";
13     ELSE "PLURAL";
14     CLOSE;
15     -->PT.HD;
16     PERSONQ(ST); GET(FLIST.HD,[N1 N2 N3])-->PT.HD;
17   EXIT;
18
19   COMMENT 'THE CLAUSE UNDERSTANDS ITS SUBJECT, SO GET IT NEXT DOOR.';
20   PPC.PPBRO->ST;
21   LOOPIF(ST.PPUNIT,NOT(EQUAL("S"))) THEN ST.PPBRO->ST; CLOSE;
22   ST.PPSON->ST;
23   LOOPIF("TENSED",NOT(ISIN(ST.PPFE))) THEN ST.PPBRO->ST; CLOSE;
24   IF ("SNG",ISIN(ST.PPFE)) THEN "SINGULAR";
25   ELSE "PLURAL";
26   CLOSE;
27   -->PT.HD;
28   HD(WHICH([1 2 3],LAMBDA; ISIN(ST.PPFE); END))-->PT.HD;
29   GET(PT.HD.HD,[N1 N2 N3])->PT.HD.HD;
30   END;
31
```

```
[MAKESIF]                          [ 20.4 11 MAY 1974]
DTRACK 217     CREATED    20.38    11  5 1974


  1   FUNCTION MAKESIF DATA PPC;
  2   COMMENT 'THIS FUNCTION MAKES A COUNTERFACTUAL HYPOTHETICAL CLAUSE:`
  3           'IF DATA IS JUST ONE CLAUSE, MAKES "PL COULD HAVE...",`
  4           'OTHERWISE "IF PL HAD..., PL WOULD HAVE...".';
  5   VARS FEATURES ADJUNCT;
  6   IF DATA.LENGTH=1 THEN
  7     MAKES(DATA,[MODAL MODALPOS PERFECTIV],NIL,PPC); .ERASE;
  8   ELSE;
  9     COMMENT 'THE SECOND CASE ABOVE.`
 10             'FIRST DEFINE THE IF... CLAUSE.';
 11     [DEPENDENT BOUND IFBOUND FINITE PAST PERFECTIVE]->FEATURES;
 12     ([%DATA.HD,DATA.TL.HD%]<>DATA.TL.TL.TL.TL)->ADJUNCT;
 13     [%[%ADJUNCT,FEATURES%]%]->ADJUNCT;
 14     COMMENT 'NOW DEFINE THE MAIN CLAUSE.';
 15     NIL->DATA.TL.TL.TL.TL;
 16     [MODAL MODALFUT PERFECTIV ENVIRADJ IFADJ]->FEATURES;
 17     MAKES(DATA,FEATURES,ADJUNCT,PPC); .ERASE;
 18   CLOSE;
 19   END;
 20
```

```
[MAKEVG]                          [ 20.47 11 MAY 1974]
DTRACK 217     CREATED    20.44    11   5 1974


    VARS VGFTORDR;
    [
     [[WILL CAN BE HAVE BEGOING] > [ASPECTUAL MODAL] >
      [GRAMMATICAL LEXICAL] > [REMOTE PRESNT PRESPART PASTPART TO] >
      [INFINITI PARTICIP TENSED] >VG > [1 2 3] > [SNG PL]
     ]
    ]->VGFTORDR;


    FUNCTION MAKEVG PPC;
    VARS PT F;
    "VG"->PPC.PPUNIT;

    COMMENT '1. GET FEATURE LIST INTO HANDY ORDER.';
    SB2(PPC.PPFE,VGFTORDR)->PPC.PPFE;

    COMMENT '2. USE THE FEATURES TO MAKE THE VERB.';
    PPC.PPFE->PT;
    LOOPIF PT.GIVEN THEN
      PT.DESTPAIR->PT->F;
      IF F.ISNUMBER THEN;
      ELSEIF F.IDENTPROPS=0 AND F.VALOF.ISFUNC THEN APPLY(F.VALOF);
      CLOSE;
    CLOSE;
    END;

```

```
[MAKEWORD]                    [ 20.41 11 MAY 1974]
DTRACK 217    CREATED   20.38   11  5 1974

 1   FUNCTION MAKEWORD PPC;
 2   COMMENT 'DEDUCES FROM FEATURES IN PPC.PPFE WHICH CLOSED-CLASS'
 3           'DICTIONARY WORD GOES IN PPC.PPSELF.';
 4   VARS ST; PPC.PPFE->ST;
 5   "WORD"->PPC.PPUNIT;
 6   IF ("PREPOSITION",ISIN(ST)) THEN
 7     WHICH([BETWEEN BY OF TO WITH],LAMBDA X; X,ISIN(ST); END)
 8     ->PPC.PPSELF;
 9   ELSEIF ("ADVERB",ISIN(ST)) THEN
10     WHICH([FIRST JUST YET],LAMBDA; ISIN(ST); END)->PPC.PPSELF;
11     COMMENT 'AMALGAMATE "NOT" WITH PRECEDING GRAMMATICAL VB.';
12     IF ("NEGATIVE",ISIN(ST)) THEN
13       IF ("GRAMMATICAL",ISIN(PPC.PPBRO.PPFE)) THEN
14         SUFFIX(PPC.PPBRO.PPSELF.HD,"NT")->PPC.PPBRO.PPSELF.HD;
15       ELSE "NOT"-->PPC.PPSELF;
16       CLOSE;
17     CLOSE;
18   ELSEIF ("BINDER",ISIN(ST)) THEN
19     WHICH([ALTHOUGH IF],LAMBDA; ISIN(ST); END)->PPC.PPSELF;
20   ELSEIF ("PRONOUN",ISIN(ST)) THEN PRONOMINALISE(PPC);
21   ELSEIF ("ARTICLE",ISIN(ST)) THEN
22     IF ("POSSESS",ISIN(ST)) THEN
23       HD(WHICH([1 2 3],LAMBDA X; (X,ISIN(ST)); END))->ST;
24       GET(ST,[MY YOUR [HIS HER ITS]])->ST;
25       IF ST.ISLIST THEN
26         IF ("INAN",ISIN(PPC.PPFE)) THEN ST.TL.TL.HD;
27         ELSEIF ("FEM",ISIN(PPC.PPFE)) THEN ST.TL.HD;
28         ELSE ST.HD;
29         CLOSE;
30         ->ST;
31       CLOSE;
32       [%ST%]->PPC.PPSELF;
33     ELSE GET((("DEFINITE",ISIN(ST))+1),[A THE])::NIL->PPC.PPSELF;
34     CLOSE;
35   ELSEIF ("NOUN",ISIN(ST)) THEN
36     [%NEWREF%]->PPC.PPSELF;
37     IF (NEWREF,ISIN([MIDDLE END])) THEN "SQUARES"->NEWREF;
38     ELSEIF ("COMMON",ISIN(ST)) AND ("SG",ISIN(ST)) THEN
39       CHOPOFF(PPC.PPSELF.HD,"S")->PPC.PPSELF.HD;
40     CLOSE;
41   ELSEIF ("S",ISIN(ST)) THEN
42     NIL->PPC.PPSELF; PPC.PPBRO->ST;
43     LOOPIF ST.PTDL THEN ->ST; CLOSE;
44     ST.PPSELF->ST;
45     LOOPIF ST.TL.NULL.NOT THEN ST.TL->ST; CLOSE;
46     (ST.HD.DATALIST<>[32 51]).MAKESTR->ST.HD;
47     PPC.PPBRO->PPC.PPDAD.PPSON;
48   ELSEIF ("NUMBER",ISIN(PPC.PPFE)) THEN
49     NUMBERWORD(REF.LENGTH)-->PPC.PPSELF;
50   CLOSE;
51   END;
52
```

```
[NFEATFNS]                          [ 20.34 11 MAY 1974]
DTRACK 217     CREATED    20.31    11  5 1974


  1    FUNCTION SG REF;
  2    IF REF.LENGTH=1 THEN "SG"; ELSE "PL"; CLOSE;
  3    -->FLIST;
  4    END;
  5
  6    FUNCTION PROPERNOUN Q; (Q.HD,ISA(PERSON))+("SG",ISIN(FLIST))=2; END;
  7
  8    FUNCTION ANIMATEQ REF;
  9    IF EVERY(REF,LAMBDA; ISA(PERSON);END) THEN "AN"; ELSE "INAN"; CLOSE;
 10     -->FLIST;
 11    IF (REF.HD,EQUAL(CLARIBEL)) THEN "FEM"-->FLIST; CLOSE;
 12    END;
 13
 14    FUNCTION PERSONQ REF;
 15    IF NO(REF,LAMBDA; ISA(PLAYERS);END) THEN 3;
 16    ELSEIF (PROTEUS,ISIN(REF)) THEN 1;
 17    ELSEIF SOME(REF,LOGGEDON) THEN 2;
 18    ELSE 3; CLOSE;
 19    -->FLIST;
 20    END;
 21
```

```
[NGFNS]                    [ 20.5 11 MAY 1974]
DTRACK 217    CREATED   20.46   11  5 1974


  1   FUNCTION NUMBERWORD N;
  2   GET(N,[ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE]);
  3   END;
  4
  5   FUNCTION PRONOUN REF;
  6   VARS HEAD PT SAVE;
  7   IF (1,ISIN(FLIST)) OR (2,ISIN(FLIST)) THEN
  8     "PROREF"-->FLIST; 1;
  9   EXIT;
 10   IF ("RELATIVE",ISIN(FLIST)) OR (REF,ISIN(CANLIST)) THEN
 11     ("PROREF"::("RELATIVE"::FLIST))->FLIST; 1;
 12   EXIT;
 13
 14   COMMENT 'SEARCH PREVREF FOR A SUITABLE ANTECEDENT.';
 15   PREVREF->PT; NIL->SAVE;
 16   LOOPIF PT.GIVEN THEN
 17     PT.DESTPAIR->PT ->HEAD;
 18     IF HEAD=NIL THEN;
 19     ELSEIF (REF,EQUAL(HEAD.HD)) THEN
 20       "PROREF"-->FLIST;
 21       IF INTERSECT([%"AN","NOUN"%],HEAD.TL.TL.HD).NULL AND
 22       SOME(WHICH(SAVE,GIVEN),LAMBDA; .HD.HD,ISA(LOCATION); END) OR
 23       SOME(CANLIST,LAMBDA; .HD,ISA(LOCATION); END) THEN
 24         ("DEICTIC"::("REMOTE"::FLIST))->FLIST;
 25         ("DEF"::("NONDET"::FLIST))->FLIST;
 26       CLOSE;
 27       1; RETURN;
 28     CLOSE;
 29   CLOSE;
 30   0;
 31   END;
 32
 33   FUNCTION PROSTRING REF;
 34   VARS PT SW;
 35   1->SW;
 36   COMMENT '1. LOOK THROUGH CANLIST.';
 37   CANLIST->PT;
 38   LOOPIF PT.GIVEN THEN
 39     IF PT.HD.ISLIST AND (REF.HD,ISA(PT.HD.HD.TYPE.HD)) THEN
 40       1,RETURN;
 41     ELSEIF (PT.HD.HD,ISA(LOCATION)) AND (REF.HD,ISA(LOCATION)) THEN
 42       0; RETURN;
 43     CLOSE;
 44     PT.TL->PT;
 45   CLOSE;
 46
 47   COMMENT '2. LOOK THROUGH PREVREF.';
 48   PREVREF->PT;
 49   LOOPIF PT.GIVEN THEN
 50     IF PT.HD.NULL THEN 0->SW;
 51     ELSEIF (PT.HD.TL.HD.VALOF,ISIN(TYPE(HD(REF)))) THEN
 52       IF ("PROSTRING",NOT(ISIN(PT.HD.TL.TL.HD))) OR SW THEN
 53         1; RETURN;
 54       CLOSE;
 55     ELSE (PT.HD.HD.HD,ISA(PLAYERS)) ->SW;
 56     CLOSE;
```

```
57      PT.TL->PT;
58    CLOSE;
59    0;
60    END;
61
62
63    [DETERMINE]↑↑↑
64    [CHECKMENDE]↑↑↑
65
66    FUNCTION PRONOMINALISE PPC;
67    VARS ST ST1 ST2;
68    PPC.PPFE->ST;
69    IF ("DEMONSTRATIVE",ISIN(ST)) THEN [THAT]; GOTO OUT; CLOSE;
70    IF ("RELATIVE",ISIN(ST)) THEN [WHICH]; GOTO OUT; CLOSE;
71    IF ("INDEF",ISIN(ST)) THEN
72      GET((("PL",ISIN(ST))+1),[ONE ONES])::NIL; GOTO OUT;
73    CLOSE;
74    WHICH([1 2 3],LAMBDA; ISIN(ST); END),HD->ST1;
75    IF ("POSSESS",ISIN(ST)) THEN
76      [[MINE YOURS [HIS HERS ITS]] [OURS YOURS THEIRS]];
77    ELSE
78      [[[I ME] [YOU YOU] [[HE HIM] [SHE HER] [IT IT]]]
79       [[WE US] [YOU YOU] [THEY THEM]]];
80    CLOSE;
81    ->ST2;
82    GET((("PL",ISIN(ST))+1),ST2)->ST2;
83    GET(ST1,ST2)->ST2;
84    IF ST2.ISLIST AND ST2.LENGTH=3 THEN
85       IF ("INAN",ISIN(ST)) THEN ST2.TL.TL.HD;
86     ELSEIF ("FEM",ISIN(ST)) THEN ST2.TL.HD;
87     ELSE ST2.HD;
88     CLOSE;
89     ->ST2;
90    CLOSE;
91    IF ("POSSESS",ISIN(ST)) THEN [%ST2%];
92    ELSE [%GET((("ACCUSATIVE",ISIN(ST))+1),ST2)%];
93    CLOSE;
94    OUT: ->PPC.PPSELF;
95    END;
96
97
98
99    FUNCTION GENITIVE NGOUT;
100   NGOUT.REV->NGOUT;
101   IF ("N",ISIN(FLIST)) THEN (NGOUT.HD.DATALIST<>[32 51]).MAKESTR;
102   ELSEIF (1,ISIN(FLIST)) THEN "MY";
103   ELSEIF (2,ISIN(FLIST)) THEN "YOUR";
104   ELSEIF ("AN",ISIN(FLIST)) THEN
105    IF ("FEM",ISIN(FLIST)) THEN "HER"; ELSE "HIS"; CLOSE;
106   ELSE "ITS";
107   CLOSE;
108   ->NGOUT.HD; REV(NGOUT);
109   END;
110
```

```
[NG RULES]                          [ 20.43 11 MAY 1974]
DTRACK 217    CREATED    20.39    11  5 1974


 1   VARS NGFTRRS;
 2   [
 3    [NG            [+HEAD]]
 4    [CLAUSE        [+CLAUSE]]
 5    [SG            [+SINGULAR = HEAD]]
 6    [PL            [+PLURAL = HEAD]]
 7    [DEF           [+DEF]]
 8    [NOTDEF        [+INDEF]]
 9    [INDEF         [+INDEF]]
10    [NUMBERED      [+CARD]]
11    [MODIFIED      [+MODIFIER]]
12    [ADJGQUAL      [+QUALIFIER = +ADJG]]
13    [CLAUSEQUAL    [+QUALIFIER = +RCLAUSE]]
14    [PREPGQUAL     [+QUALIFIER = +PREPG]]
15    [DANGLINGP     [+DANGLINGP = PREPG]]
16    [NOUN          [+CLASS]]
17    [PROPERNOUN    [+NAME]]
18    [PRONOUN       [+PRON]]
19    [PROSTRING     [+ANAPHORS = PRON]]
20    [PROREF        [+ANAPHORR = PRON]]
21    [1             [+ADRESSER]]
22    [2             [+ADRESSEE]]
23    [3             [+REFEREE]]
24    [AN            [+LIFE = HEAD]]
25    [FEM           [+FEM = HEAD]]
26    [DETERMINED    [+DET]]
27    [DEICTIC       [+DEICTIC = ANAPHORR]]
28    [RELATIVE      [+REL = PRON]]
29    [REMOTE        [+REMOTE = DEICTIC]]
30    [NOMPOSS       [+NG = DET]]
31    [ADJPOSS       [+ADJG = DET]]
32    [OBJECT        [+ACCUSATIVE = HEAD]]
33    [GENITIVE      [+GENMKR]]
34    [POSSESS       [+DET IFF [-DET PROREF]]
35                   [+GENITIVE = DET IFF [-PROREF]]
36                   [+GENITIVE = ANAPHORR IFF [PROREF]]]
37   ] -> NGFTRRS;
38
39
40
41   VARS NGSBRS1 NGSBRS2;
42   [
43    [+DET IFF [DEF] AND [-DET ANAPHORR NAME]]
44    [+DET IFF [INDEF] AND [ANAPHORS] AND [MODIFIER]]
45    [+DET IFF [INDEF] AND [-ANAPHORS DET CARD PLURAL]]
46    [HEAD = [CLAUSE PRON NAME CLASS MODIFIER CARD]]
47    [HEAD = [ADRESSER ADRESSEE REFEREE]]
48    [+OFOBJ = PREPG IFF [PREPG] AND [PREPG = QUALIFIER]]
49   ]->NGSBRS1;
50
51   [
52    [INDEF = [DET HEAD]]
53    [DEF = [DET ANAPHORR NAME]]
54    [DET > CARD > MODIFIER > [CLAUSE CLASS NAME PRON] > ADJG
55        > PREPG > RCLAUSE > GENMKR]
56   ] -> NGSBRS2;
```

```
57
58
59
60   VARS NGFURRS;
61   [
62    [ACCUSATIVE     [+ACCUSATIVE]]
63    [ADRESSER       [+1]]
64    [ADRESSEE       [+2]]
65    [ADJG           [+ADJG]
66                    [+COMPLEX IFF [QUALIFIER]]]
67    [ANAPHORR       [+DEFINITE IFF [DEF]]
68                    [+INDEF IFF [-DEF]]
69                    [+INAN IFF [-LIFE]]]
70    [ANAPHORS       [+INDEF]]
71    [CARD           [+WORD] [+NUMBER]]
72    [CLASS          [+WORD] [+NOUN] [+COMMON]]
73    [CLAUSE         [+CLAUSE] [+DEPENDENT]]
74    [DEF            [+DEFINITE IFF [DET]]]
75    [DEICTIC        [+DEMONSTRATIVE]]
76    [DET            [+WORD IFF [-HEAD DEICTIC NG ADJG GENITIVE]]
77                    [+ARTICLE IFF [-HEAD DEICTIC NG ADJG]]]
78    [GENITIVE       [+GENITIVE IFF [NG]]
79                    [+POSSESS IFF [-NG]]]
80    [GENMKR         [+WORD] [+S]]
81    [INDEF          [+INDEF IFF [DET]]]
82    [LIFE           [+AN]]
83    [MODIFIER       [+ADJG] [+SIMPLE]]
84    [NAME           [+WORD] [+NOUN] [+PROPER]]
85    [NG             [+NG]]
86    [OFOBJ          [+OF]]
87    [PLURAL         [+PL]]
88    [PREPG          [+PREPG] [+DANGLINGP IFF [DANGLINGP]]]
89    [PRON           [+WORD] [+PRONOUN]]
90    [RCLAUSE        [+CLAUSE] [+DEPENDENT]]
91    [REFEREE        [+3]]
92    [REL            [+RELATIVE]]
93    [REMOTE         [+REMOTE]]
94    [SINGULAR       [+SG]]
95   ]->NGFURRS;
96
```

```
[ODDFNS]                          [ 20.34 11 MAY 1974]
DTRACK 217     CREATED    20.31    11  5 1974


    1   FUNCTION GIVEN Q; IF Q.ISLIST AND NOT(Q.NULL) THEN 1; ELSE 0; CLOSE; E
;
    2
    3   [SETFNS]↑↑↑
    4
    5   COMPILE(LIBRARY([RANDOM]));
    6   INTOF(POPTIME/10)->RANSEED;
    7
    8   FUNCTION CHOICEOF L;
    9   GET((INTOF(.RANDOM*(L.LENGTH))+1),L);
   10   END;
   11
   12   FUNCTION READTO ITEM => LIST;
   13   VARS X; NIL->LIST;
   14   L: .ITEMREAD->X;
   15   IF X=ITEM THEN LIST.REV->LIST; EXIT;
   16   X::LIST->LIST; GOTO L;
   17   END;
   18
   19   FUNCTION APPTESTL L F;
   20   LOOPIF NOT(L=NIL) THEN;
   21    IF F(L.HD) THEN EXIT;
   22    L.TL->L;
   23   CLOSE;
   24   END;
   25
   26
   27   FUNCTION MAKESTR L => STR;
   28   INITC(LENGTH(L))->STR;
   29   VARS I; 1->I;
   30   LOOPIF L.GIVEN THEN;
   31    L.DEST->L->SUBSCRC(I,STR);
   32    I+1->I;
   33   CLOSE;
   34   END;
   35
   36   FUNCTION LOGGEDON N;
   37   IF LOGGEDON.FNPROPS.TL.NULL THEN POPUSER.VALOF;
   38   ELSE LOGGEDON.FNPROPS.TL.HD;
   39   CLOSE;
   40   EQUAL(N);
   41   END;
   42   LAMBDA X; [%X%]->LOGGEDON.FNPROPS.TL; END->UPDATER(LOGGEDON);
   43
   44
```

```
[ODDFNS 2]                    [ 20.34 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974


 1   FUNCTION ROLELIST;"INITL","ACT","GOAL"; END;
 2
 3   FUNCTION MOVENO; (1+HOWMANY(£SQUARES,OCCUPIED)); END;
 4
 5   FUNCTION NIMMT PL SQ;
 6   IF SAMEDATA(SQ,ENTITY) THEN SQ.CONTENTS.HD->SQ; CLOSE;
 7   [%SQ,PL,.MOVENO%]-> £(SQ.COORDS.BOARDV);
 8   END;
 9
10   FUNCTION DISPLAY;
11   VARS L HEAD SPACES SW; 0->SW;;
12   £HISTORY -> L;
13   2.NL; 3.SP;
14   L.HD.HD->HEAD; PR(NNAME(HEAD));
15   ([%NNAME(HEAD).DESTWORD%].LENGTH-1)->SPACES;
16    5.SP; PR(NNAME(L.TL.HD.HD));
17   1.NL; (INTOF(SPACES/2))->SPACES;
18   LOOPIF L.GIVEN THEN;
19    L.DEST->L; .TL.HD->HEAD;
20    IF SAMEDATA(HEAD,ENTITY) THEN £HEAD.HD->HEAD; CLOSE;
21    IF NOT(SW) THEN SP(SPACES+2); HEAD.PR; 1->SW;
22    ELSE SP(SPACES+5); HEAD.PR; 1.NL; 0->SW;
23    CLOSE;
24   CLOSE;
25   2.NL;
26   END;
27
28
29   FUNCTION OPPONENT PL;
30   VARS X; CONTENTS(HISTORY)->X;
31   IF LENGTH(X)=<1 THEN "-";
32   ELSEIF X.HD.HD=PL THEN X.TL.HD.HD;
33   ELSE X.HD.HD; CLOSE;
34   END;
35
36   FUNCTION LASTWO ST;
37   VARS ST1;
38   IF ST THEN NONOP > ->ST; (.MOVENO-3);
39   ELSE NONOP < ->ST; 3; CLOSE;
40   ->ST1;
41   IF ST1<0 THEN NIL; EXIT;
42   WHICH(£SQUARES, LAMBDA X;
43    IF X.OCCUPIED AND ST(£X.TL.TL.HD,ST1) THEN 1; ELSE 0; CLOSE;
44     END);
45   END;
46
47   FUNCTION TAKEFN X;
48   COMMENT 'THIS FUNCTION IS APPLIED IN EQCL TO RESTRICT XEQCL IN`
49           'THE CASE OF A SQUARE WHICH IS THE OBJECT OF "TAKE".`
50           'ONLY A VACANT SQUARE CAN BE TAKEN, SO NON-VACANT ONES`
51           'CAN BE EXCLUDED FROM XEQCL.`;
52   IF X.GIVEN AND (X.HD,ISA(SQUARES)) THEN WHICH(X,VACANT);
53   ELSE X;
54   CLOSE;
55   END;
56
```

```
[PREPG RULES]                          [ 20.47 11 MAY 1974]
DTRACK 217     CREATED    20.44    11   5 1974


    1    VARS PREPGFTR PREPGSB PREPGFUR;
    2    [
    3     [PREPG            [+NOM IFF [-COORD DANGLINGP]]]
    4     [RELATIVE         [+RELATIVE = NOM IFF [-COORD]]]
    5     [BETWEEN          [+PREP = +BETWEEN]]
    6     [BY               [+PREP = +BY]]
    7     [OF               [+PREP = +OF]]
    8     [TO               [+PREP = +TO]]
    9     [WITH             [+PREP = +WITH]]
   10    ]->PREPGFTR;
   11
   12    [
   13     [PREP > NOM]
   14    ]->PREPGSB;
   15
   16    [
   17     [NOM              [+NG] [+OBJECT]]
   18     [RELATIVE         [+RELATIVE]]
   19     [PREP             [+WORD] [+PREPOSITION]]
   20     [BETWEEN          [+BETWEEN]]
   21     [BY               [+BY]]
   22     [OF               [+OF]]
   23     [TO               [+TO]]
   24     [WITH             [+WITH]]
   25    ]->PREPGFUR;
   26
   27
```

```
[RDMOVES]                          [ 20.32 11 MAY 1974]
DTRACK 217     CREATED    20.30    11  5 1974


  1    FUNCTION READMOVES;
  2    VARS X Y Z;
  3    NIL->Z;;
  4    1.NL; 'NAME FIRST PLAYER: '.PRSTRING; .ITEMREAD->X;
  5    1.NL; 'ENTER MOVES AS LIST: '.PRSTRING;
  6    X::(.LISTREAD)->X;
  7    1.NL; 'SECOND PLAYER: '.PRSTRING; .ITEMREAD->Y;
  8    1.NL; 'MOVES: '.PRSTRING; Y::(.LISTREAD)->Y;
  9    APPLIST(X.TL,LAMBDA XX;
 10     [%X.HD.VALOF,XX%]-->Z;;
 11     IF Y.TL.NULL THEN;
 12     ELSE [%Y.HD.VALOF,Y.TL.HD%]-->Z; Y.TL.TL->Y.TL; CLOSE;
 13     END));
 14    IF Y.TL.GIVEN THEN [%Y.HD.VALOF,Y.TL.HD%]-->Z;CLOSE;
 15    Z.REV-> £HISTORY;
 16    END;
 17
```

```
[SB FNS]                    [ 20.4 11 MAY 1974]
DTRACK 217    CREATED   20.38   11  5 1974


 1   FUNCTION SB1 INPUT RULES => INPUT;
 2   COMMENT 'SB1 DOES ADDITION AND CONFLATION SB RULES FOR CLAUSE.';
 3   VARS CR CSR ST;
 4
 5   L1: IF RULES.NULL THEN EXIT;
 6     RULES.DEST->RULES->CR;
 7
 8   L2: COMMENT '1. GET AND TEST CONDITION, IF ANY, ON RULE.';
 9     IF GETCOND(CR) THEN ->ST;
10       TESTCOND(ST,INPUT)->ST->INPUT;
11       IF ST=FALSE THEN; GOTO L1; CLOSE;
12     CLOSE;
13
14   L3: COMMENT 'CONDITION, IF ANY, IS MET SO DO JOB.';
15     PERFORM(CR,INPUT)->INPUT;
16   GOTO L1;
17   END;
18
19
20   FUNCTION BUNDLE ST ST1 => ST;
21   IF ST.ISLIST.NOT THEN ST::NIL->ST;
22   ELSEIF ST.GIVEN AND ST.HD.ISLIST THEN ST.HD->ST; CLOSE;
23   IF ST1.ISLIST.NOT THEN [%ST1%]->ST1;
24   ELSEIF  ST1.GIVEN AND ST1.HD.ISLIST THEN ST1.HD->ST1; CLOSE;
25   (ST<>ST1)->ST;
26   IF ST.LENGTH>1 THEN ST::NIL->ST; CLOSE;
27   END;
28
29   FUNCTION SB2 INPUT RULES => OUTPUT;
30   COMMENT 'EXECUTES SEQUENCE SB RULES.';
31   VARS PT CR ST STPT ST1 ST2 EQSW EQSW2 BFSW;
32   NIL->OUTPUT; 0->EQSW; 0->EQSW2; NIL->ST; ST->STPT;
33
34   L1: COMMENT 'GET NEXT RULE, IF ANY, PROVIDED INPUT IS NOT NULL.';
35     IF ST.GIVEN THEN
36       IF EQSW AND EQSW2 THEN 0->EQSW2; GOTO L7;
37       ELSEIF NOT(EQSW) THEN
38         IF PT=0 THEN OUTPUT->PT; CLOSE;
39         GOTO L6;
40       ELSE ST.HD-->INPUT;
41       CLOSE;
42     CLOSE;
43     0->EQSW2;
44     IF RULES.NULL OR INPUT.NULL THEN EXIT;
45     RULES.DEST->RULES->CR;
46     NIL->ST; ST->STPT; 0->PT; NIL->ST2;
47     (CR.TL.HD="=")->EQSW;
48
49   L2: COMMENT 'CHECK WHETHER AT END OF CURRENT RULE.';
50     IF CR.NULL THEN; GOTO L1; CLOSE;
51
52   L3: COMMENT 'TRAVERSE CURRENT RULE.';
53     CR.DEST->CR->ST1;
54     IF ST1.ISLIST THEN
55       IF ST1.GIVEN THEN ST1.DEST->ST2->ST1;
56       ELSE; GOTO L2;
```

```
57        CLOSE;
58      ELSEIF ST1=">" THEN
59        0->EQSW; 0->EQSW2; GOTO L3;
60      ELSEIF ST1="=" THEN
61        1->EQSW; GOTO L3;
62      CLOSE;
63
64    L4: COMMENT 'ST1 CONTAINS A WORD.';
65      IF BFETCH(INPUT,ST1) THEN ->ST1; 1->BFSW;
66      ELSEIF BFIND(OUTPUT,ST1) THEN ->PT; 0->BFSW;
67      ELSE
68        IF ST2.NULL THEN; GOTO L2;
69        ELSE ST2.DEST->ST2->ST1; GOTO L4;
70        CLOSE;
71      CLOSE;
72
73    L5: COMMENT 'THE WORD HAS BEEN FOUND - DEAL WITH IT.'
74                'ADD ST1 TO ST IF NECESSARY.';
75      IF EQSW AND NOT(BFSW) THEN
76        1->EQSW2; NIL->ST1;
77      CLOSE;
78      IF BFSW OR EQSW THEN
79        IF EQSW THEN STPT; ELSEIF STPT.NULL THEN NIL; ELSE STPT.TL; CLOSE;
80        BUNDLE(ST1);
81        IF ST.NULL OR EQSW THEN ->ST; ST->STPT;
82        ELSE ->STPT.TL; STPT.TL->STPT;
83        CLOSE;
84      CLOSE;
85
86    COMMENT 'NOW ADD ST1 TO OUTPUT IF NECESSARY.';
87      IF BFSW THEN
88        IF NOT(EQSW) THEN
89      L6:
90          IF PT=0 THEN;
91          ELSE
92         IF PT.NULL THEN ST->PT;
93              IF OUTPUT.NULL THEN ST->OUTPUT; CLOSE;
94            ELSE PT.TL->STPT.TL; ST->PT.TL; ST->PT;
95            CLOSE;
96            NIL->ST; ST->STPT;
97          CLOSE;
98        CLOSE;
99      ELSE
100     L7:
101       IF EQSW THEN
102        BUNDLE(PT.HD,ST.HD)->ST;
103         IF EQUAL(OUTPUT.HD,PT.HD) THEN ST.HD->OUTPUT.HD;
104         ELSE ST.HD->PT.HD;
105         CLOSE;
106       ELSEIF ST.GIVEN THEN
107        PT.COPYLIST->STPT.TL; ST.HD->PT.HD; ST.TL->PT.TL;STPT.TL->PT;
108       CLOSE;
109       NIL->ST; ST->STPT;
110     CLOSE;
111     GOTO L2;
112   END;
113
114
```

```
[SETFNS]                    [ 20.34 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974


 1   FUNCTION MEMBER Q L;
 2   L: IF L.NULL THEN 0;
 3      ELSEIF EQUAL(Q,L.HD) THEN 1;
 4      ELSE L.TL->L; GOTO L; CLOSE;
 5   END;
 6
 7   FUNCTION INTERSECT S1 S2;
 8   IF S1.NULL THEN NIL EXIT;
 9   S1.HD; INTERSECT(S1.TL,S2);
10   IF MEMBER(S1.HD,S2) THEN ::; ->S2; ELSE ->S2;.ERASE; CLOSE;
11   S2;
12   END;
13
14   FUNCTION COMPLEMENT SUBSET SET => SET;
15   VARS ST;
16   IF SET.NULL THEN EXIT;
17   SET.REV->ST;
18   NIL->SET;
19   LOOPIF ST.GIVEN THEN ST.DEST->ST; ::SET->SET;
20    IF (SET.HD,MEMBER(SUBSET)) THEN SET.TL->SET; CLOSE;
21   CLOSE;
22   END;
23
24
25   FUNCTION CONDENSE L => L;
26   IF L.NULL THEN EXIT;
27   L.HD::(L.TL.CONDENSE)->L;
28   IF (L.HD,MEMBER(L.TL)) THEN L.TL->L; CLOSE;
29   END;
30
31   FUNCTION LISTDELETE LIST ITEM => ST;
32   COMMENT 'ITEM MAY BE A LIST OF ITEMS FOR DELETION, OR AN ITEM.';
33   NIL->ST;
34   IF LIST.NULL THEN EXIT;
35   LOOPIF LIST.NULL.NOT THEN
36     IF ITEM.ISLIST.NOT AND EQUAL(ITEM,LIST.HD) THEN;
37     ELSEIF ITEM.ISLIST AND (LIST.HD,MEMBER(ITEM)) THEN;
38     ELSE (LIST.HD::ST)->ST;
39     CLOSE;
40     LIST.TL->LIST;
41   CLOSE;
42   ST.REV->ST;
43   END;;
44
45   FUNCTION GET NTH LIST;
46   IF NTH=1 THEN LIST.HD; EXIT;
47   GET(NTH-1,LIST.TL);
48   END;
49
50   FUNCTION NCREV LIST;
51   VARS X Y;
52   IF LIST.NULL THEN LIST; EXIT;
53   NIL->X; LIST->Y;
54   LOOP:
55    Y.TL; X->Y.TL; Y->X; ->Y;
56    IF Y.NULL THEN X; ELSE; GOTO LOOP; CLOSE;
```

```
57    END;
58    FUNCTION COINCIDE SET1 SET2;
59    BOOL AND(LENGTH(SET1)=LENGTH(SET2),
60             LENGTH(SET1)=LENGTH(INTERSECT(SET1,SET2)));
61    END;
62
63
```

```
[SET UP]                          [ 20.41 11 MAY 1974]
DTRACK 217    CREATED   20.38   11  5 1974


  1   [ODDFNS]↑↑↑
  2   [MACROS]↑↑↑
  3   [BFETCH]↑↑↑
  4   [CLAUSE RULES]↑↑↑
  5   [FTR FNS]↑↑↑
  6   [SB FNS]↑↑↑
  7   VARS CPL ENVIRON LAST NODESTOCK ROOT WHDEPTH;
  8   0->DEBUG; IDENTFN->ENVIRON; NIL->NODESTOCK;
  9   VARS TESTSW TESTSW1 TESTSW2 TESTSW3 TESTSW4;
 10   0->TESTSW; 0->TESTSW1; 0->TESTSW2; 0->TESTSW3; 0->TESTSW4;
 11   0->TESTSW3;
 12   [TREEFNS]↑↑↑
 13   [ENTITY]↑↑↑
 14   [HCLHFNS]↑↑↑
 15   [ODDFNS 2]↑↑↑
 16   [V SETUP]↑↑↑
 17   [EQCL]↑↑↑
 18   [UPDATE1]↑↑↑
 19   [CONJUNCTION TABLE]↑↑↑
 20   FUNCTION SOMEHOW; END; FUNCTION DRAW; END; FUNCTION WIN; END;
 21   VARS OPPONENT;
 22   [DESIGN0]↑↑↑
 23   [TREE DISPLAY FNS]↑↑↑
 24   [RDMOVES]↑↑↑
 25   [DESIGN0]↑↑↑
 26   [DESIGN1]↑↑↑
 27   [MAKEWORD]↑↑↑
 28   [MAKES]↑↑↑
 29   [MAKESFN1]↑↑↑
 30   [CLAUSE FUNCTION FNS]↑↑↑
 31   [MAKESIF]↑↑↑
 32   [LOCFNS]↑↑↑
 33   [LRRP]↑↑↑
 34   [CHECKMEN]↑↑↑
 35   [SPOUT]↑↑↑
 36   [SPIELEN]↑↑↑
 37   [NG RULES]↑↑↑
 38   [PREPG RULES]↑↑↑
 39   [DETERMINE]↑↑↑
 40   [MAKENG]↑↑↑
 41   [NGFNS]↑↑↑
 42   [NFEATFNS]↑↑↑
 43   [MAKEPNG]↑↑↑
 44   [V SETUP]↑↑↑
 45   [MAKEVG]↑↑↑
 46   [VG FNS]↑↑↑
 47   [LOCDESCR FNS]↑↑↑
 48   [LOCDESCR FNS 2]↑↑↑
 49   [LOCD10]↑↑↑
 50   [LOCDESCR]↑↑↑
 51   [MAKEADJG]↑↑↑
 52   [SPATREL]↑↑↑
 53   [MAKERELNG]↑↑↑
 54
```

```
[SPIELEN]                          [ 20.34 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974


 1   FUNCTION SQVAL SQ;
 2   IF SQ.OCCUPIED THEN £SQ.TL.HD.NNAME.DATALIST.HD;
 3   ELSEIF NSW THEN £SQ.HD;
 4   ELSE 16;
 5   CLOSE;
 6   END;
 7
 8   FUNCTION BPRINT NSW;
 9   VARS I; 1->I;
10   L1:
11    '    .    .'.PRSTRING; 1.NL;
12   L2:
13    1.SP; [%"SQUARE",I,"GOON"%].POPVAL.SQVAL.CUCHAROUT;
14    IF ((I//3).ERASE) THEN 1.SP; ".".PR; I+1->I; GOTO L2; CLOSE;
15   L3:
16    1.NL;
17    IF I<9 THEN '.....'.PRSTRING; 1.NL; 1+I->I; GOTO L1; CLOSE;
18   2.NL;
19   END;
20
21
22   [GAMEFNS]↑↑↑
23   FUNCTION FORK PL PL1;
24   VARS ST;
25   WHICH(£LINES,OPEN)->ST;
26   WHICH(ST,
27    LAMBDA X; WHICH(£X,OCCUPIED).HD.CONTENTS.TL.HD,EQUAL(PL); END)
28   ->ST;
29   IF ST.NULL THEN; ST; EXIT;
30   WHICH(£SQUARES,
31    LAMBDA X;
32    IF X.VACANT AND HOWMANY(ST,LAMBDA Y; X,ISIN(£Y); END)>1 THEN 1;
33    ELSE 0; CLOSE;
34   END);
35   END;
36
37   FUNCTION THREATEN PL PL1;
38   VARS ST X; NIL->X;
39   WHICH(
40    WHICH(£LINES,OPEN),
41    LAMBDA X; WHICH(£X,OCCUPIED)->X; £(X.HD).TL.HD,EQUAL(PL); END)
42   ->ST;
43   IF ST.NULL THEN ST; EXIT;
44   APPLIST(ST, LAMBDA Y; WHICH(£Y,VACANT)->Y;
45    (Y<>X)->X; END);
46   X;
47   END;
48
49   FUNCTION SOMEHOW F;
50   VARS ST; APPLY(F)->ST;
51   IF ST.GIVEN THEN ST,1; ELSE 0; CLOSE;
52   END;
53
54   FUNCTION PLAN => OUT;
55   VARS ST;
56   MAPLIST(WHICH(£SQUARES,VACANT),LAMBDA X; ASSESS([%CPL,X%]); END)
```

```
57    ->ST;
58    APPTESTLIST([WIN DRAW BLOCK FORK THREATEN TAKE],
59     LAMBDA F;
60      WHICH(ST,LAMBDA X; F,ISIN(X); END)->OUT;
61      IF OUT.GIVEN THEN CHOICEOF(OUT)->OUT; 1;
62      ELSE 0; CLOSE;
63     END);
64    END:
65
66    [%WIN,FORK,THREATEN,DRAW,BLOCK%]->ASSESSLIST;
67
68
69    FUNCTION PLAY;
70    VARS ST ST1 OCPL;
71    NIL-> £HISTORY; NIL->CANLIST; NIL->PREVREF;
72    IF LOGGEDON.FNPROPS.TL.GIVEN THEN LOGGEDON.FNPROPS.TL.HD;
73    ELSE POPUSER.VALOF; CLOSE;
74    ->ST;
75    .RESET; 1->PAGELINE; 1->LINEPOS;
76    CONSNODE("S",NIL,NIL,NIL,NIL,NIL)->ROOT;
77    IF CHOICEOF([1 0]) THEN PROTEUS->CPL; ST->OCPL;
78      MAKES([%PROTEUS,NIL,CHOICEOF([START BEGIN]),[%HD(£GAME)%]%],
79            [PRESENT IMMINENT EXT DESCR OP],NIL,ROOT);
80      .ERASE;
81    ELSE ST->CPL; PROTEUS->OCPL;
82      IF CHOICEOF([0 1]) THEN ("USTANDA"::NIL); ELSE NIL; CLOSE;
83      ->ST;
84      MAKES([%CPL,NIL,CHOICEOF([START BEGIN]),[%HD(£GAME)%]%],
85       (ST<>[PRESENT MODAL MODALFUT INDEP INDIC INTERROG EXT DESCR OP]),
86            NIL,ROOT);
87      .ERASE;
88    CLOSE;
89    ROOT.SENTPR; 1.NL;
90    NIL->ROOT.PPSON; NIL->PREVREF; 1->LINEPOS;
91
92    L1:
93     IF (CPL,EQUAL(PROTEUS)) THEN
94      .PLAN->ST; NIMMT(CPL,ST.TL.HD); ST--> £HISTORY;
95    ELSE IF .MOVENO>1 THEN 0.BPRINT; CLOSE;
96     'YOUR MOVE'.PRSTRING;
97     .NUMBERREAD->ST; NIMMT(CPL,ST); [%CPL,ST%]--> £HISTORY;
98     CLOSE;
99    IF SAMEDATA(ST,ENTITY) THEN £ST.HD->ST; CLOSE;
100   CPL,OCPL->CPL->OCPL;
101
102    IF SOME(WHICH(£LINES,OCCUPIED),
103     LAMBDA X;
104      £X->X;
105      ALL(X,LAMBDA Y; £Y.TL.HD,EQUAL(OCPL); END);
106     END) THEN
107     IF CHOICEOF([0 1]) THEN [%HD(£GAME)%],"GTRANS";
108     ELSE NIL,"GINTRANS";
109     CLOSE;
110     ->ST1->ST;
111     MAKES([%OCPL,NIL,"WIN",ST%],(ST1::[PRESENT PERFECTIV]),NIL,ROOT);
112     .ERASE;
113    ELSEIF EVERY(£LINES,DEAD) THEN
114      MAKES([%NIL,NIL,"DRAW",[%HD(£GAME)%]%],[PRESENT PERFECTIV],
115            NIL,ROOT); .ERASE;
116    ELSE; GOTO L1;
```

```
117     CLOSE;
118     1.NL; ROOT.SENTPR; 2.NL; 0.BPRINT; 2.NL; REV(£HISTORY)-> £HISTORY;
119    END;
120
```

```
[SUFFIX]                          [ 20.32 11 MAY 1974]
DTRACK 217     CREATED    20.30    11  5 1974


 1   FUNCTION SUFFIX WORD LETTERS;
 2   [%WORD.DESTWORD%].REV.TL->WORD;
 3   [%LETTERS.DESTWORD%]->LETTERS;
 4   IF HD OF WORD = 37 THEN
 5    IF HD OF LETTERS = 37 THEN LETTERS.TL->LETTERS;
 6    ELSEIF VOWEL(HD OF LETTERS) THEN
 7    IF LENGTH OF WORD>2 THEN WORD.TL->WORD; CLOSE;
 8    CLOSE;
 9   ELSEIF HD OF WORD = 40 THEN
10    IF CONSONT(HD OF LETTERS) THEN 37-->WORD; CLOSE;
11   ELSEIF (HD OF WORD,ISIN([45 46])) THEN
12    IF VOWEL(HD OF LETTERS) AND NOT(LETTERS.HD=37)
13     THEN HD OF WORD -->WORD; CLOSE;
14   CLOSE;
15   ((LETTERS.REV.TL)<>WORD)->WORD;
16   LENGTH OF WORD -->WORD;
17   IF WORD.HD>8 THEN MAKESTR(WORD.TL.REV);
18    ELSE SPIT(WORD); CLOSE;
19   END;
20
21   FUNCTION CHOPOFF WORD LETTERS;
22   WORD.DATALIST.REV->WORD;
23   LETTERS.DATALIST.REV->LETTERS;
24   LOOPIF LETTERS.GIVEN AND EQUAL(LETTERS.HD,WORD.HD) THEN;
25     LETTERS.TL->LETTERS; WORD.TL->WORD; CLOSE;
26    SPIT(LENGTH OF WORD ::WORD);
27   END;
28
```

```
[TREE DISPLAY FNS]                          [ 20.35 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974

 1   FUNCTION NODEPR LINEPOS ST => LINEPOS;
 2   VARS WORDLENGTH F;
 3
 4   L1:
 5    IF ST.ISLIST AND ST.GIVEN THEN;
 6    ELSEIF ST.ISFUNC OR ST=NIL THEN RETURN;
 7    ELSE ST.PR; RETURN;
 8    CLOSE;
 9    IF SAMEDATA(ST.HD,' `) THEN PRSTRING,DATALENGTH(ST.HD);
10    ELSEIF ST.HD.ISWORD THEN PR, LENGTH([%ST.HD.DESTWORD%])-1;
11    CLOSE;
12    ->WORDLENGTH->F;
13    IF (LINEPOS+WORDLENGTH)>68 AND (ST.HD,NOT(ISIN([.,]))) THEN
14    1->LINEPOS; 2.NL;
15    ELSEIF INTERSECT([, .],ST).NULL THEN 1.SP;
16    CLOSE;
17    ST.HD; APPLY(F); ST.TL->ST;
18    (LINEPOS+WORDLENGTH+1)->LINEPOS;
19   GOTO L1;
20   END;
21
22   FUNCTION SENTPR PPC;
23   VARS ST;
24   IF PPC.PTLT THEN ->PPC; ELSE EXIT;
25   L1: PPC.PPSELF->ST;
26    NODEPR(LINEPOS,ST)->LINEPOS;
27    IF PTRT(PPC) THEN ->PPC; GOTO L1; CLOSE;
28   END;
29
30   VARS PAGELINE; 0->PAGELINE;
31
32   FUNCTION TREEPR PPC;
33   VARS PSV NLIST NLISTSV DADLIST PLIST NPLIST SPACES SAVELPOS PLISTSV;
34   NIL->NLIST; NIL->DADLIST; NIL->PLIST; NIL->NPLIST; NIL->PLISTSV;
35   LINEPOS->SAVELPOS; 1->LINEPOS;
36   PPC->PSV;
37   IF (PAGELINE+(3*(DEPTH(PPC))))>60 THEN
38    64.CUCHAROUT; 1->PAGELINE; CLOSE;
39   2.NL;
40
41   COMMENT '1 - PRINT LEAVES RECORDING PRINTING POSITIONS.';
42   PPC.PTLT.ERASE->PPC;
43   L1:
44    LINEPOS-->NPLIST; PPC-->DADLIST;
45    NODEPR(LINEPOS,PPC.PPSELF)->LINEPOS;
46    IF NOT(PPC.PTRT) THEN; GOTO L11; CLOSE;
47    ->PPC;
48    LINEPOS-(NPLIST.HD)->SPACES;
49    IF SPACES=<5 THEN 5-SPACES; ELSE 0; CLOSE;
50    ->SPACES;
51    SP(SPACES); SPACES+LINEPOS->LINEPOS;
52   GOTO L1;
53
54   L11:DADLIST.REV->DADLIST; DADLIST->NLIST; NLIST->NLISTSV;
55
56   L2: COMMENT 'PRINT FIRST HALF BRANCH.';
```

```
 57    NPLIST.REV->NPLIST;
 58    NPLIST->PLIST; PLIST->PLISTSV;
 59    1.NL: 1->LINEPOS;1.SP;
 60    LOOPIF PLIST.GIVEN THEN
 61     SP(PLIST.HD-LINEPOS); PR("."); PLIST.HD+1->LINEPOS;
 62     PLIST.TL->PLIST;
 63    CLOSE;
 64
 65   L3: COMMENT 'PRINT SECOND HALF BRANCH,';
 66    COMMENT '* IN DADLIST MARKS YOUNGER SON OF UPCOMING FATHER.';
 67    1.NL; 1.SP; 1->LINEPOS;
 68    NIL->NLIST; NIL->NPLIST;
 69    L31:
 70     IF DADLIST.NULL THEN NLIST.REV->NLIST; GOTO L4; CLOSE;
 71     DADLIST.DEST->DADLIST->PPC;
 72     IF SAMEDATA(PPC,ROOT) THEN
 73      SP(PLISTSV.HD-LINEPOS);
 74      PR(".");    PPC-->NLIST; PLISTSV.HD-->NPLIST;
 75     ELSE LOOPIF LINEPOS =<PLISTSV.HD THEN
 76           PR("-"); LINEPOS+1->LINEPOS;
 77          CLOSE;
 78     CLOSE;
 79     PLISTSV.DEST->PLISTSV; ->LINEPOS; 1+LINEPOS->LINEPOS;
 80     GOTO L31;
 81
 82   L4: COMMENT 'PRINT THE NEXT LEVEL OF NODES.';
 83     NIL->DADLIST;
 84    NPLIST.REV->PLIST; NIL->NPLIST;
 85    1.NL; 1.SP; 1->LINEPOS;
 86    NLIST->NLISTSV;
 87    LOOPIF NLIST.GIVEN THEN NLIST.DEST->NLIST->PPC;
 88     IF (PPC,EQUAL(PSV.PPDAD)) THEN; GOTO L5; CLOSE;
 89     SP(PLIST.HD-LINEPOS);
 90      PLIST.DEST->PLIST->LINEPOS;
 91      LINEPOS-->NPLIST;
 92     IF NLIST.GIVEN AND SOME(NLIST,LAMBDA; BELOW(PPC);END) THEN
 93      ".".PR; LINEPOS+1->LINEPOS;
 94     ELSEIF DADLIST.GIVEN AND
 95       SAMEDATA(DADLIST.HD,ROOT) AND BELOW(DADLIST.HD,PPC.PPDAD) THEN
 96       ".".PR; LINEPOS+1->LINEPOS;
 97     ELSE PPC.PPUNIT.PR;
 98      ([%PPC.PPUNIT.DESTWORD%].LENGTH-1+LINEPOS)->LINEPOS;
 99      IF PPC.PTU THEN ->PPC;
100      ELSE; GOTO L41;
101      CLOSE;
102     CLOSE;
103    IF (PPC,ISIN(DADLIST)) THEN "*";
104    ELSE PPC;
105    CLOSE;
106     -->DADLIST;
107    L41:
108    CLOSE;
109
110    IF DADLIST.GIVEN THEN
111     NLISTSV->NLIST; DADLIST.REV->DADLIST; GOTO L2; CLOSE;
112
113   L5:SAVELPOS->LINEPOS;
114   2.NL;
115   END;
116
```

```
 1  VARS CONSNODE MKND PPUNIT PPFE PPSON PPDAD PPBRO PPSELF PPC;
 2  RECORDFNS("NODE",[0 0 0 0 0 0])
 3  ->PPBRO -> PPSON -> PPDAD -> PPSELF -> PPFE -> PPUNIT;
 4  .ERASE; -> MKND;
 5
 6  FUNCTION CONSNODE UNIT FE SELF DAD SON BRO;
 7  VARS ST;
 8  IF NODESTOCK.GIVEN THEN NODESTOCK.DEST->NODESTOCK->ST;
 9   UNIT->ST.PPUNIT; FE->ST.PPFE; SELF->ST.PPSELF;
10   DAD->ST.PPDAD; SON->ST.PPSON; BRO->ST.PPBRO; ST;
11  ELSE MKND(UNIT,FE,SELF,DAD,SON,BRO); CLOSE;
12  END;
13
14  FUNCTION PTDL PPC; PPC.PPSON->PPC;
15  IF PPC=NIL THEN 0; ELSE PPC,1; CLOSE;
16  END;
17
18  FUNCTION PTPV PPC; PPC.PPBRO->PPC;
19  IF PPC=NIL THEN 0; ELSE PPC,1; CLOSE;
20  END;
21
22  FUNCTION PTDF PPC;
23  VARS X;
24  IF PPC.PTDL THEN ->PPC; ELSE 0; EXIT;
25  LOOP:
26   IF PPC.PPBRO=NIL THEN PPC,1; EXIT;
27   PPC.PPBRO->PPC; GOTO LOOP;
28  END;
29
30  FUNCTION PTU PPC;
31  PPC.PPDAD->PPC;
32  IF PPC=NIL THEN 0; ELSE PPC,1; CLOSE;
33  END;
34
35  FUNCTION PTYBRO PPC;
36  VARS X;
37  IF PPC.PTU THEN ->X; X.PPSON->X;
38  ELSE 0; EXIT;
39  IF PPC=X THEN 0; EXIT;
40  LOOP:
41   IF X=NIL THEN 0; RETURN;
42  ELSEIF X.PPBRO=PPC THEN X,1;
43   ELSE X.PPBRO->X; GOTO LOOP; CLOSE;
44  END;
45
46  FUNCTION PTLMOST PPC;
47  VARS X;
48  LOOP:
49   PPC.PPBRO->X;
50   IF X=NIL THEN PPC; EXIT;
51   X->PPC; GOTO LOOP;
52  END;
53
54  FUNCTION ADDTO PPC;
55  IF PPC.PPDAD.PTDL THEN -> PPC.PPBRO; CLOSE;
56  PPC->PPC.PPDAD.PPSON;
```

```
 57   END;
 58
 59   FUNCTION TQ PPC; PPC.PPSON=NIL; END;
 60
 61   FUNCTION PTRT PPC;
 62   COMMENT 'RETURNS THEN NEXT RIGHT TERMINAL NODE,1, OR 0.';
 63   IF PPC.PTYBRO THEN ->PPC;
 64    LOOP: IF PPC.TQ THEN PPC,1; EXIT;
 65          IF PPC.PTDF THEN ->PPC; GOTO LOOP;
 66          ELSE 0; EXIT;
 67   ELSEIF PPC.PTU THEN ->PPC; PPC.PTRT;
 68   ELSE 0;
 69   CLOSE;
 70   END;
 71
 72   FUNCTION PTLT PPC;
 73   COMMENT 'RETURNS PPC OR LEFTMOST TERMINAL DESCENDANT,1, OR 0.';
 74   IF PPC.TQ THEN PPC, 1;
 75   ELSEIF PPC.PTDF THEN ->PPC; PPC.PTLT;
 76   ELSE 0;
 77   CLOSE;
 78   END;
 79
 80   FUNCTION BELOW PPC PPC1;
 81   IF (PPC,EQUAL(PPC1)) THEN 0; EXIT;
 82   LOOPIF PPC.PTU THEN ->PPC;
 83    IF (PPC,EQUAL(PPC1)) THEN 1; EXIT;
 84   CLOSE;
 85   0;
 86   END;
 87
 88   FUNCTION DEPTH PPC;
 89   VARS ST;1->ST;
 90   LOOPIF PPC.PTDL THEN ->PPC; ST+1->ST; CLOSE;
 91   ST;
 92   END;
 93
 94   FUNCTION PTS F Q PPC;
 95   COMMENT 'SEARCHES FROM PPC F-WARDS UNTIL A Q NODE.';
 96   LOOPIF PPC.F THEN ->PPC;
 97    IF PPC.PPUNIT=Q THEN PPC, 1; EXIT;
 98   CLOSE;
 99   0;
100   END;
101
102    FUNCTION PTKINDLE PPC;
103   VARS ST; NIL->ST;
104   L0:
105    IF PPC=NIL THEN EXIT;
106    IF PPC.PTDL THEN .PTKINDLE; NIL->PPC.PPSON; CLOSE;
107    PPC.PPBRO->ST; PPC::NODESTOCK->NODESTOCK; ST->PPC;
108   GOTO L0;
109   END;
110
```

```
[SPATREL]                    [ 20.35 11 MAY 1974]
DTRACK 217    CREATED   20.31   11  5 1974

 1   FUNCTION SPATIALRELATION XEQCL COMPLEQCL REFPOINT;
 2   IF ([%REFPOINT%],ISIN(CANLIST)) THEN; 0; EXIT;
 3   NONMAP(XEQCL,COMPLEQCL,[%ADJACENT,OPPOSITE%]);
 4   END;
 5
 6   FUNCTION SPATPART XEQCL REFPOINT PPC => OK;
 7   COMMENT 'TRIES DESCRIPTION OF SQUARES IN XEQCL AS PART OF REFPOINT.';
 8   VARS COMPLEQCL ST ST1 ST2;
 9
10   IF (XEQCL.HD,ISA(SQUARES)).NOT OR NOT(REFPOINT,ISA(EDGES)) THEN
11     0->OK;
12   EXIT;
13   INTERSECT(XEQCL,£REFPOINT)->XEQCL;
14   IF XEQCL.NULL THEN 0->OK; EXIT;
15   INTERSECT(COMPLEMENT(XEQCL,XEQCL.HD.TYPE.HD.CONTENTS),£REFPOINT)
16   ->COMPLEQCL;
17   IF NONMAP(XEQCL,COMPLEQCL,[%MIDDLE,END1%]) THEN
18     .HD->ST1; 1->OK;
19   ELSEIF NONMAP(XEQCL,NIL,[%MIDDLE,END1%]) THEN
20     .HD->ST1;
21     REF->PPC.PPSELF;
22     LOCD2(PPC)->OK; NIL->PPC.PPDAD.PPSON;
23     IF NOT(OK) THEN
24       LOCD4(PPC)->OK; NIL->PPC.PPDAD.PPSON;
25     CLOSE;
26   CLOSE;
27   IF NOT(OK) THEN EXIT;
28
29   L1: COMMENT 'NOW IDENTIFY THE REFPOINT IF POSSIBLE.';
30   PPC.PPDAD.PPFE->ST2;
31   IF PPC.PPUNIT="ADJG" THEN
32     CONSNODE(NIL,NIL,NIL,PPC.PPDAD,NIL,NIL)->ST;
33   ELSE PPC->ST;
34   CLOSE;
35   IF ("PREPREL",ISIN(PPC.PPDAD.PPDAD.PPFE)) THEN; GOTO L2; CLOSE;
36
37   [PREPG OF].COPYLIST->ST.PPFE; [%REFPOINT%]->ST.PPSELF;
38   IF ("DANGLINGP",ISIN(ST2)) THEN
39     "DANGLINGP"-->ST.PPFE;
40   CLOSE;
41   MAKEPNG(ST)->OK; NIL->PPC.PPDAD.PPSON;
42   IF NOT(OK) THEN EXIT;
43
44   COMMENT 'AMEND THE FEATURE SET OF THE PARENT NG.';
45   ("QUALIFIED"::("PREPGQUAL"::ST2))->ST2;
46   L2: LISTDELETE(ST2,[NOUN PRONOUN PROREF PROSTRING])->ST2;
47   IF PPC.PPUNIT="ADJG" THEN "MODIFIED"-->ST2; CLOSE;
48   ("NOUN"::ST2)->PPC.PPDAD.PPFE;
49   COMMENT 'REPLACE NOUN IN DOMINATING NG.';
50   ST1->NEWREF; XEQCL->PPC.PPDAD.PPSELF;
51   COMMENT 'ORGANISE COMPONENTS FOR CONSTRUCTING NG.';
52   IF PPC.PPUNIT="ADJG" THEN ST-->SAVESET;
53   ELSE ST->PPC;
54   CLOSE;
55   END;
56
```

```
[SPOUT]                          [ 20.36 11 MAY 1974]
DTRACK 217    CREATED   20.32   11  5 1974


 1   FUNCTION SPOUT;
 2   VARS ADJUNCT CPL CANLIST CANLIST2 FEATURES MOVES PPC REST ST X;
 3   £HISTORY->REST; IDENTFN->ENVIRON; 0->LAST;
 4   NIL->PREVREF; .RESET; NIL->FEATURES; NIL->ADJUNCT;
 5   0->LINEPOS; NIL->CANLIST;
 6   CONSNODE("S",NIL,NIL,NIL,NIL,NIL)->PPC;
 7   PPC->ROOT;
 8
 9   L1:
10    LOOPIF .STACKLENGTH>0 THEN .ERASE; CLOSE;
11    IF PPC.PTDL THEN .ERASE;
12      IF TESTSW2 THEN 1.NL; 1->LINEPOS; 0.BPRINT; CLOSE;
13      PPC.SENTPR; PPC.PTKINDLE; NIL-->PREVREF;
14      CONSNODE("S",NIL,NIL,NIL,NIL,NIL)->ROOT; ROOT->PPC;
15    CLOSE;
16    IF REST.NULL THEN
17     IF LAST THEN 2.NL; EXIT;
18     COMMENT 'THE GAME WAS UNFINISHED.';
19     MAKES([%CPL,NIL,CHOICEOF([END FINISH]),[%HD(£GAME)%]%],
20           [EXT DESCR MID PRESENT PERFECTIV PRADJ NEGATIVE],NIL,PPC);
21     ->LAST;
22     ADDTO(CONSNODE(NIL,NIL,[%"."%],ROOT,NIL,NIL));
23     GOTO L1;
24    CLOSE;
25
26     COMMENT 'GROUP NEXT BLOCK OF MOVES.';
27     .DESIGN1.REV->ST;
28   COMMENT 'ROOT.PPSELF NOW CONTAINS A LIST OF THE GROUPED MOVES.';
29     PPC.PPSELF->MOVES;
30     IF MOVES.LENGTH>1 THEN [COORD SEG CONJUNC]->ROOT.PPFE; CLOSE;
31   COMMENT 'GROW NEXT LEVEL.';
32     LOOPIF ST.GIVEN THEN ST.DEST->ST->X;
33       0->WHDEPTH;
34       IF (X,ISIN([HOWEVER AND BUT SO , .])) THEN;
35         COMMENT 'ADD THIS WORD DIRECTLY.';
36         ADDTO(CONSNODE("WORD",NIL,[%X%],PPC,NIL,NIL));
37
38       ELSEIF (X="IF") THEN;
39         COMMENT 'MAKE A CLAUSE TO MENTION A HYPOTHETICAL MOVE.';
40         ADDTO(CONSNODE("S",NIL,NIL,PPC,NIL,NIL));
41         MAKESIF(ST.HD,PPC.PPSON);
42         ST.TL->ST;
43
44       ELSE;
45         COMMENT 'MAKE A STANDARD CLAUSE AND ADD IT.';
46         IF ("COORD",ISIN(PPC.PPFE)) THEN
47           ADDTO(CONSNODE("S",NIL,NIL,PPC,NIL,NIL)); PPC.PPSON->PPC;
48         CLOSE;
49         MAKES(X,FEATURES,ADJUNCT,PPC); .ERASE; ROOT->PPC;
50         IF X.HD.ISLIST THEN X.TL.TL.HD; ELSE X.TL.HD; CLOSE;
51         ->X;
52         IF EQUAL(X,MOVES.HD.TL.HD) THEN
53           NIMMT(MOVES.HD.HD,MOVES.HD.TL.HD); MOVES.TL->MOVES;
54         CLOSE;
55       CLOSE;
56     CLOSE;
```

```
57
58    NIL->EQCL.FNPROPS.TL;
59    NIL->ROOT.PPFE;
60    NIL->CHECKDE.FNPROPS.TL;
61    NIL->LRRP.FNPROPS.TL;
62    GOTO L1;
63    END:
64
```

```
[UPDATE1]                          [ 20.35 11 MAY 1974]
DTRACK 217     CREATED    20.31    11   5 1974


    VARS PREVREF;
    FUNCTION UPDATE1 ENTRY;
    VARS CURRENT CFTS EFTS SAVE;
    NIL->SAVE;
    PREVREF.DEST->PREVREF->CURRENT;
    INTERSECT([1 3 SG PL AN INAN],
     CURRENT.TL.TL.HD)->CFTS;

    LOOPIF PREVREF.GIVEN THEN PREVREF.DEST->PREVREF->ENTRY;
     IF ENTRY.GIVEN.NOT THEN
      IF (NIL,ISIN(SAVE)) THEN NIL->PREVREF; CLOSE;
      NIL-->SAVE; GOTO L0;
     CLOSE;
     ENTRY.TL.TL.HD->EFTS;
     IF (ENTRY.HD,EQUAL(CURRENT.HD)) THEN;
     ELSEIF (ENTRY.TL.HD,EQUAL(CURRENT.TL.HD)) THEN;
     ELSEIF ALL(CFTS,LAMBDA;ISIN(EFTS);END) AND
       ("PROSTRING",NOT(ISIN(CURRENT.TL.TL.HD))) AND
       SOME((CURRENT.HD<>ENTRY.HD),LAMBDA;ISA(LOCATION);END) THEN;
     ELSE ENTRY-->SAVE;
     CLOSE;
     L0:
    CLOSE;
    (CURRENT::(REV(SAVE)))->PREVREF;
    END;
```

```
 1  FUNCTION GRAMMATICAL;
 2  (PPC.PPFE.HD::NIL)->PPC.PPSELF;
 3  IF PPC.PPSELF.HD="BEGOING" THEN [BE GOING]->PPC.PPSELF; CLOSE;
 4  END;
 5
 6  FUNCTION PRESPART; SUFFIX(PPC.PPSELF.HD,"ING")->PPC.PPSELF.HD; END;
 7
 8  FUNCTION PASTPART;
 9  VARS ST;
10  FETCH(PPC.PPSELF.HD)->ST;
11  IF ST.TL.HD="WEAK" THEN SUFFIX(PPC.PPSELF.HD,"ED");
12  ELSE ST.TL.TL.TL.HD;
13  CLOSE;
14  ->PPC.PPSELF.HD;
15  END;
16
17  FUNCTION TENSE SW PPC;
18  COMMENT 'SELECTS THE CORRECT TENSE, PERSON, & NUMBER OF VERB.';
19  VARS ST P N;
20  FETCH(PPC.PPSELF.HD)->ST;
21
22  COMMENT '1. GET PERSON & NUMBER OF SUBJECT.';
23  IF ("SNG",ISIN(PPC.PPFE)) THEN "SNG"; ELSE "PL"; CLOSE; ->N;
24  HD(WHICH([1 2 3],LAMBDA; ISIN(PPC.PPFE); END))->P;
25  IF SW THEN; GOTO L1; CLOSE;
26
27  COMMENT 'THE VERB IS PRESENT TENSE';
28  IF ST.TL.HD.ISWORD OR ST.TL.HD.NULL THEN
29    IF P=3 THEN SUFFIX(PPC.PPSELF.HD,"S"); CLOSE;
30  ELSE
31    IF N="PL" THEN 2; ELSE P; CLOSE;
32    GET(ST.TL.HD);
33  CLOSE;
34  ->PPC.PPSELF.HD;
35  RETURN;
36
37  L1: COMMENT 'THE VERB IS PAST TENSE.';
38  IF ST.TL.HD="WEAK" THEN SUFFIX(PPC.PPSELF.HD,"ED");
39  ELSE ST.TL.TL.HD->ST;
40    IF ST.ISLIST THEN
41      IF N="PL" THEN 2; ELSE P; CLOSE;
42      GET(ST);
43    ELSE ST;
44    CLOSE;
45  CLOSE;
46  ->PPC.PPSELF.HD;
47  END;
48
49
50  FUNCTION TO; "TO"-->PPC.PPSELF; END;
51
52  FUNCTION REMOTE; TENSE(1,PPC); END;
53
54  FUNCTION PRESNT; TENSE(0,PPC); END;
55
```

```
[V SETUP]                          [ 20.4 11 MAY 1974]
DTRACK 217     CREATED    20.38    11   5 1974


 1    VARS PARADIGM;
 2    [ [BE [AM ARE IS] [WAS WERE WAS] BEEN [0 INTRANS]]
 3      [BEGIN NIL BEGAN BEGUN [1 TRANS INTRANS]]
 4      [BLOCK WEAK [2 TRANS]]
 5      [CAN [CAN CAN CAN] COULD NIL [0 INTRANS]]
 6      [COMPLETE WEAK [2 TRANS]]
 7      [DO [DO DO DOES] DID DONE [2 TRANS INTRANS]]
 8      [DRAW NIL DREW DRAWN [2 TRANS INTRANS]]
 9      [END WEAK [1 TRANS INTRANS]]
10      [FINISH WEAK [1 TRANS INTRANS]]
11      [FORK WEAK [2 TRANS INTRANS]]
12      [HAVE [HAVE HAVE HAS] HAD HAD [2 TRANS]]
13      [LOSE NIL LOST [2 TRANS]]
14      [MARCH WEAK [1]]
15      [START WEAK [1 TRANS INTRANS]]
16      [TAKE NIL TOOK TAKEN [2 TRANS]]
17      [THREATEN WEAK [2 TRANS]]
18      [WASH WEAK [2]]
19      [WILL [SHALL WILL WILL] WOULD WILL [0 INTRANS]]
20      [WIN NIL WON WON [2 TRANS]]
21    ]->PARADIGM;
22
23    FUNCTION FETCH V;
24    APPLIST(PARADIGM,LAMBDA X; IF X.HD=V THEN X; CLOSE; END);
25    END;
26
27    FUNCTION SPIT V;CONSWORD(APPLIST(V,REV,IDENTFN));  END;
28
29    FUNCTION VOWEL Q; MEMBER(Q,[33 37 41 47 53]); END;
30
31    FUNCTION CONSONT Q; NOT(VOWEL(Q)); END;
32
33    FUNCTION PROPERTIES V; V.FETCH.REV.HD; END;
34
35    FUNCTION VERBTYPE V; HD(PROPERTIES(V)); END;
36
37    [SUFFIX]↑↑↑
38
```

APPENDIX B

APPENDIX B

A worked example of sentence construction

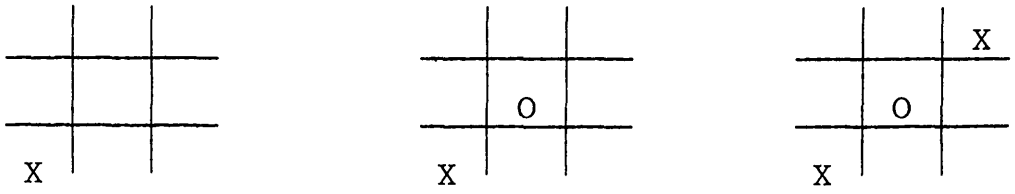
Note

In the example below procedure names will be written in capitals and underlined, as SPOUT. Entry to a procedure will be indicated by >, thus >SPOUT, and exit from a procedure by <, as <SPOUT. Text explaining what a particular procedure does will be inset, as will the entry to or exit from a procedure called within another procedure. Thus depth of inset will be a guide to the depth of function-call we are at.

Variable names will be written in capitals, as REST, and if their contents is given, it will appear after a colon. Thus REST:{<Proteus> take} indicates a variable called REST whose current contents is a list of two elements, an internal representation of the player called Proteus and the quoted word 'take'.

Workings

Suppose the game to be described was unfinished, comprising just the moves diagrammed on the next page:



These three moves are input and stored in a variable HISTORY thus;

HISTORY: {{<Proteus> 7} {<ACD> 5} {<Proteus> 3}}

The program can now be told to SPOUT.

>SPOUT:

The procedure first initialises global variables, and then sets a pointer REST to the part of HISTORY yet to be described; at first REST points to the start of HISTORY. After these preliminaries, SPOUT calls in specialists to design the next sentence, in this case the first one.

>DESIGN1:

The operation of the DESIGN1 procedure was described in detail in chapter 6, section 3; in the present case DESIGN1 calls ASSESS to evaluate each move, and places the result of ASSESS in a variable ROOT, which is going to be the root node of the surface structure tree.

>ASSESS:

REST: {{<Proteus> 7} { <ACD> 5} {<Proteus> 3}}

<ASSESS

ROOT: {{<Proteus> <square 7> start <game> take <square 7>}}

DESIGN1 reviews progress and calls for the

evaluation of another move.

<ASSESS

REST: {{<ACD> 5} {<Proteus> 3}}

<ASSESS

ROOT: {{<Proteus> <square 7> start <game> take
<square 7>}
{<ACD> <square 5> take <square 5>}}

DESIGN1 again reviews progress, and invokes ASSESS a third time.

>ASSESS:

REST: {<Proteus> 3}

<ASSESS

ROOT: {{<Proteus> <square 7> start <game> take
<square 7>}
{<ACD> <square 5> take <square 5>}
{<Proteus> <square 3> take <square 3>}}

This completes the assessment of the first three moves. The assessment is simple, since moves 2 and 3 have no tactical points of interest to them. DESIGN1 has meantime been assembling the sentence design in a temporary variable. The design comprises the information put in ROOT, together with conjunctions and punctuation. The design thus largely duplicates the information in ROOT. The redundancy is motivated by the convenience of having just one data type, namely lists, as members of the list in ROOT. The sentence design

then is

OUT: {{<Proteus> <square 7> start <game>
take <square 7>},
{<ACD> <square 5> take <square 5>}, and
{<Proteus> <square 3> take <square 3>}.}

The final action of DESIGN1 is to undo the gedanken moves which it made after ASSESSing each move; these gedanken moves were necessary in order to establish the right situation for the ASSESSment of the next move.

<DESIGN1

Control has now returned to SPOUT. The procedure prepares to construct each element in the design made ready by DESIGN1. No further work of construction is needed for elements which are quoted words, namely punctuation marks and conjunctions; such elements are simply placed in position directly dominated by ROOT. Other elements in the design are lists. These represent clauses and are attached likewise directly to the ROOT node, but their construction is handed over to the specialist clause-maker MAKES.

All further activity of SPOUT may thus be summarised by the following sequence of procedure calls and exits:

>ADDTO:

Attaches a clause node to ROOT.

<ADDTO

>MAKES:

Constructs the clause represented in the design by {<Proteus> <square 7> start <game> take <square 7>} and attaches its constituent - structure subtree to the {Clause} node.

<MAKES:

The next element in the design is the quoted word ',' which can be set in place without further computation.

>ADDTO

Attaches a {Word} node to ROOT, with ',' at it.

<ADDTO

The remaining elements are succesively placed on the tree:

>ADDTO:

<ADDTO

>MAKES:

Constructs the clause represented by the element {<ACD> <square 5> take <square 5>}

<MAKES

>ADDTO:

Attaches a {Word} node with ',' at it.

<ADDTO

>ADDTO:

Attaches a {Word} node with 'and' at it.

<ADDTO

>MAKES:

Constructs the clause represented by the element

Proteus square 3 take square 3

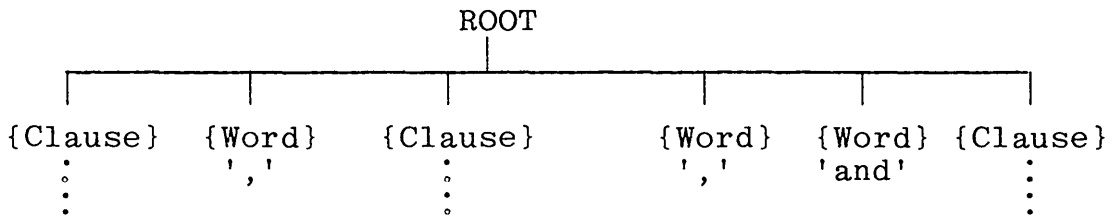
<MAKES

>ADDTO:

Attches a {Word} node with '.' at it.

<ADDTO

The function SPOUT has now organised the construction of a complete sentence by attaching nodes as follows to ROOT:



SPOUT therefore prints the words at the bottom of the constituent-structure tree whose top level is shown here, and then prepares for the next sentence. In this example the game was unfinished when it ended, so a standard comment design is selected and the comment is constructed in just the same way as the sentence just examined. SPOUT then finishes.

<SPOUT

We must now consider what happened when SPOUT called MAKES. MAKES requires a clause design and a pointer to the node at which the constituent-structure tree of the clause must be attached once built. The way the procedure works

was explained in chapter 6 section 5, and is recapitulated here only very briefly.

The first clause design is

{<Proteus> <square 7> start <game> take <square 7>}

>MAKES:

The procedure first examines the design to see whether the clause needs an environment adjunct, namely a subordinate conditional or concessive clause preceding the main clause. In this example there is no environment adjunct. (If there had been such an adjunct, it would have appeared as the first element in the design, as

{{ALTHOUGH <Proteus> <square 4> block <edge> threaten <Dan >} <Dan> win <game> complete <line>}

which would be the design underlying the sentence: 'Although I blocked your edge and threatened you you won the game by completing your line.')

Next it checks whether the clause should be made coordinate. This it does according to the rules explained in chapter 6, section 5, page 227. In the present example MAKES is able to add the features {Simple} and, of course, {Clause} to the variable FEATURES;

FEATURES: {Clause Simple}

As the system has no intensive verbs, MAKES assumes the feature {Ext}, and then traverses the

remainder of the {TRANSITIVITY} net. It takes the verb 'start' from the clause design and looks it up in a dictionary. The entry is

{START WEAK {1 TRANS INTRANS}}

which tells MAKES that 'start' is a type 1 verb, which in turn means in this system that the clause containing 'start' is {Descr}. There remains the choice of {Op}, {Mid}, or {Receptive} in system 11. The present program cannot properly motivate this choice. It therefore ignores {Receptive} as being the marked choice, and selects one of the other two at random. For this example we assume that it selected {Op}. We have at this stage:

FEATURES: {Op Descr Ext Simple Clause}

MAKES now re-enters the {ADJUNCT} systems to see whether there must be an appendix adjunct, system 40. The procedure examines the clause design and finds {...take <square 7>} at the tail. This specifies the material of a clause, and since the major clause has already been marked {Simple}, this additional one must be subordinate. MAKES therefore adds {Appadj} to FEATURES, and proceeds to system 41. In this system it ignores the {Timeadj} option because the present program does not produce any clausal time adjuncts. It makes the remaining choice between {Methodadj} and {Accompadj} according to whether the main and subordinate clauses will have the same surface subject. In the present case

the main clause has been marked {Op} so we know that its subject will be the human initiator who is also the subject of the subordinate clause. MAKES therefore selects {Methodadj}, which will bring about

'I began...by taking...'

in preference to {Accompadj} which would have given

* 'I began...with (my) taking...

At this stage we have

FEATURES: {Methodadj Appadj Op Descr Ext Simple Clause}

MAKES now pre-empts certain options of the subordinate clause. In due course this clause will be constructed by a recursive call of MAKES and the current incarnation of MAKES prepares for that event by marking the subordinate clause design {Ustand Ustandi Ustanda} and {Dependent Nonfin Ing Participle}. The subordinate clause design is now established as

{{<Proteus> <square 7> take <square 7>}
{Dependent Nonfin Ing Participle Ustand Ustandi Ustanda}}

and this information is set aside in an ADJUNCT variable.

Returning now to the main clause, MAKES traverses the {MOOD} systems. If the clause has not already been marked {Present} or {Nonfin} it is assumed to be {Past}, and if not already {Dependent} or {Indep} it is assumed {Indep Indic Declar}. Thus we have

FEATURES: {Indep Indic Declar Past Methodadj Appadj Op Descr Ext Simple Clause}

The feature set is now complete. The clause Ftr rules are applied and the result is placed in

FUNSET: ((BYOBJ APPENDIX) PAST FINITE (INITR SUBJECT) ACTOR PROCESS)

After application of the SB1 and SB2 rules this becomes

FUNSET: ((SUBJECT INITR) (PROCESS FINITE PAST) (ACTOR POSTVERB) (APPENDIX BYOBJ))

On this see chapter 5, section 5, especially the worked example in subsection 5.

MAKES must now arrange for the construction of the four constituents identified by their function bundles. In each case MAKES constructs a node on the constituent structure tree, and then invokes the appropriate specialist to build the required constituent at that node. MAKES discovers which specialist will be appropriate by first applying the clause function-realisation rules to to the relevant function-bundle, and then invoking

the specialist needed for an item marked {Clause}, {Ng}, {Prepg}, or {Vg} as the case may be.

The first constituent has the function-bundle (SUBJECT INITR). In order to ensure that the constituents of a clause represent the right meanings, certain functions of the grammar have been given undercover roles as procedures in the program; MAKES has simply to invoke whichever of each function-bundle turns out to be a procedure, and the procedure selects from the clause design the corresponding element and makes it available to the constructor of the constituent. So in the first bundle INITR is found to be a procedure and is called:

>INITR:

    Makes <Proteus> available to the constructor of the first constituent of the clause.

<INITR

The function-bundle (SUBJECT INITR) was realised in the feature-set {Ng Subject}, from which MAKES determined that MAKENG was the right constructor to call. Thus, having made a node for MAKENG to work on

>ADDTO:

<ADDTO

MAKES is ready to call MAKENG.

>MAKENG:

Constructs a noun-group to represent <Proteus> and attaches the sub-tree it has made to the node supplied by MAKES.

<MAKENG

The remaining three constituents are dealt with in exactly the same way as the first. In each case MAKES invokes ADDTO, REALISE, and then respectively PROCESS and MAKEVG, ACTOR and MAKENG, and APPENDIX and MAKEPNG. We shall examine each of these specialists in turn.

Having completed construction of the clause {<Proteus> <square 7> start <game> take <square 7>}

MAKES exits to SPOUT.

<MAKES

The first constructor called by MAKES is MAKENG:

>MAKENG:

This procedure has to construct a noun-group to convey <Proteus>. Its first task is to assemble the feature-set, subject, however, to the possibility of having to construct a noun-group

constituent in course of determining a feature (see chapter 6, section 4, page 218).

MAKENG determines from the representation <Proteus> that the present noun-group is neither <Coord> nor <Clause>. It must then be <Nominal>, and a sequence of specialists are invoked to examine the representation and so to add <Sg 1 Inan> <Proref Pronoun> and finally <Def> to the feature set, which is then

<Def Proref Pronoun Sg 1 Inan Nominal Ng Subject>

The Ftr and SB rules are applied, yielding

(HEAD SINGULAR DEF PRON ANAPHORR ADRESSER)

and the FNr rules then yield

<Sg Pron Word Inan Definite 1>

which suffices to enable the word-maker to select 'I'. (The accusative form would have been marked by exception {Accusative...}.)

MAKENG therefore concludes its work by creating the required {Word} node,

>ADDTO:

<ADDTO

and invoking the word-maker to put 'I' in the node.

>MAKEWORD:

<MAKEWORD

It then terminates.

<MAKENG

After MAKENG has run, control is returned to MAKES, which then turns to the constituent whose function-bundle is (PROCESS FINITE PAST), represented in the clause design by the quoted word {...start...}. MAKES creates a new node and then applies the FNr rules to the function-bundle to obtain

{Lexical Vg Remote Tensed}.

Details of the surface subject of the verb are supplied by a specialist invoked by MAKES. This specialist obtains the person, number, and animacy of the surface subject via the subject's ordered function bundle set. It takes special steps in cases where the subject is understood, as in

'...and blocked your edge'.

In the present example the specialist adds {Singular Inan N1} to the feature-list. MAKES now invokes MAKEVG to make the verb constituent, whose feature list is given as

{Singular Inan N1 Lexical Vg Remote Tensed}.

>MAKEVG:

It is apparent that the given feature-set is adequate in itself to identify the form of 'start' which is needed in the present case. MAKEVG therefore does not seek to add any features to the feature-set, but simply constructs the required verb form with the help of the dictionary entry for 'start', which, as we have seen, is

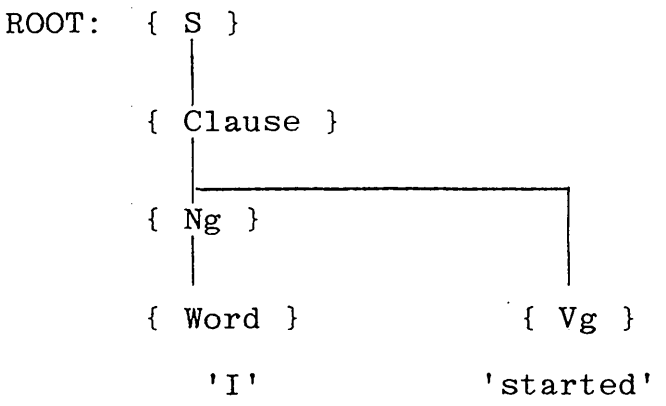
{START WEAK {1 TRANS INTRANS}}

The operation of MAKEVG was explained in chapter 6

section 10, page 278, and will not be set out again here. Having constructed 'started' and placed it at the {Vg} node, MAKEVG ends.

<MAKEVG

When MAKEVG terminates, MAKES resumes control. At this point the constituent-structure tree is:



MAKES now turns to construct the constituent which has the function-bundle (ACTOR POSTVERB). Applying the FNr rules

>REALISE:

<REALISE

it obtains the partial feature-set {Object Ng}. The semantic representation of the required noun-group is secured by calling the ACTOR procedure found in the function bundle:

>ACTOR:

<ACTOR

This yields <game> as the referent of the noun-group.

The final stage of preparation is to make the new {Ng} node:

>ADDTO:

<ADDTO

MAKES can now run MAKENG.

>MAKENG:

This procedure runs very much as it did when making the noun-group 'I'. However, in this case the feature-set finally arrived at is

{Def Noun Inan 3 Sg Nominal Ng Object}

as explained in chapter 6, section 7, especially subsections 2, 3, and 4. The feature-set is realised as

((DEF DET) (REFEREE CLASS HEAD SINGULAR ACCUSATIVE)).

There are thus two constituents of the noun-group. The feature-set of the first is realised

>REALISE:

<REALISE

as {Article Word Definite}. MAKENG creates a {Word} node and calls the word-maker to put the right word at it:

>MAKEWORD:

<MAKEWORD

The second function-bundle is realised as

{Sg 3 Common Noun Word Accusative}.

Since this is a word item the word-maker is invoked:

>MAKEWORD:

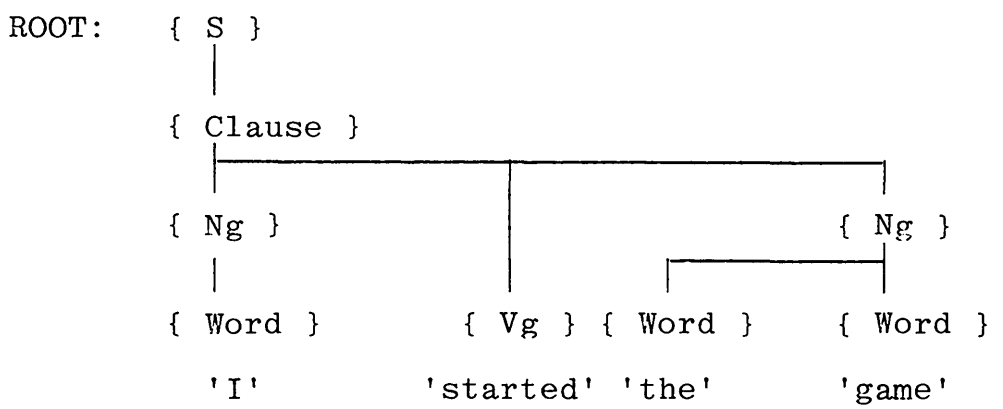
MAKEWORD examines the feature-set of the word and finds that the word is {Noun} and {Common}. The actual word needed has been placed in a variable NEWREF by MAKENG (see chapter 6, section 7.3), and MAKEWORD now takes the contents of NEWREF, here 'games' and converts the form to the singular 'game' (see chapter 6, section 10). The word 'game' can now be placed at the {Word} node, and the procedure terminates.

<MAKEWORD

MAKENG now also terminates:

<MAKENG

At this point the constituent-structure tree is:



Control has now reverted to MAKES which must arrange for the construction of the appendix constituent. We saw earlier that the design of the appendix has been pre - determined by MAKES as:

{{<Proteus> <square 7> take <square 7>}
{Dependent Nonfin Ing Participle Ustand Ustandi
Ustanda}}

The appendix constituent has the function-bundle (APPENDIX BYOBJ) which is realised in the feature-set {Prepg By}. MAKES therefore invokes MAKEPNG:

>MAKEPNG:

This procedure realises the feature-set {Prepg By} in the function bundle set ((PREP BY) (NOM)). These bundles are realised in {Word Preposition By} and {Ng Object} respectively. Each is constructed in turn:

>MAKEWORD:

The word-maker notes that the item is {Preposition} and so selects the feature {By} as the actual word required (see chapter 6, section 10).

<MAKEWORD

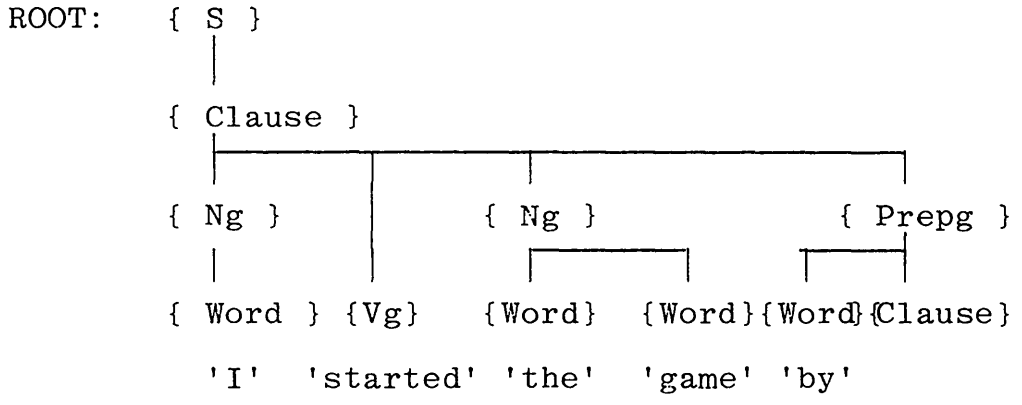
MAKENG:

This procedure is given the features {Ng Object} and at once discovers from the semantic representation, which is

{<Proteus> <square 7> take <square 7>},

that the noun-group must have the feature {Clause}. The feature-list attached to the representation includes {...Ustand Ustandi Ustanda} as features of the clause to be built, and so the Ng must be {Notdet}, as '...taking a corner', rather than '...my taking a corner' (chapter 4, section 5.2).

MAKENG now realises the Ng features derived so far, Ng Clause Notdet , as ((HEAD CLAUSE)) and then realises these functions in the features Clause Dependent . MAKENG places the semantic representation, and the union of the pre-determined feature-set with Clause Dependent , in a new node and calls the clause-maker MAKES to construct the clause. As it does so, the constituent-structure tree is at this point:



MAKES is called, and returns:

>MAKES:

<MAKES

Control comes back to MAKENG, and thence

<MAKENG

to MAKEPNG

<MAKEPNG

and thence via MAKES to SPOUT.

The first independent clause is now complete:

'I started the game by taking a corner'

and SPOUT goes on to construct the next item.

There is no great difficulty in following through the working of the program as it completes construction of the sentence, and there would be little to be gained by setting out the details in this Appendix. Enough has been said to explain what the program has to work from, and how the grammar is used.

REFERENCES

Ambler A P & Burstall R M

Question-answering and Syntax analysis

Edinburgh University, Department of Machine Intelligence

Experimental Programming Report 18 1969

Bach E & Harms R (eds)

Universals in Linguistic Theory

New York 1968

Bever T G & Langendoen D T

A dynamic model of the evolution of languages

Linguistic Inquiry 4 1971 433 - 461

Biss K O, Chien R T, & Stahl F A

A data structure for cognitive information retrieval

Coordinated Science Laboratory, University of Illinois

Report R-493 1970 (See also Schultz 1970)

Bobrow D G

Syntactic Theory in computer implementations

in Automated Language Processing, ed. Borko H

New York Wiley 1970

Bobrow D G

Natural language input for a computer problem-solving program

in: Semantic Information Processing, ed. Minsky M

Cambridge, Mass. M.I.T. 1968

Breslaw P I

Experiments with a semantic network

Edinburgh University, Department of Machine Intelligence

mimeo MIP-R63 1969

Bruce B

A model for temporal reference and its application in a question-answering program

Artificial Intelligence 3 1972

Charniak E

Jack and Janet in search of a theory of knowledge

Proceedings of the 3rd International Joint Conference on Artificial Intelligence 1973

Chomsky N

Syntactic Structures

The Hague Mouton 1957

Remarks on nominalisation

in: Studies on Semantics in Generative Grammar

The Hague Mouton 1972 (Janua Linguarum 107)

Colby K M & Smith D C

Dialogues between humans and an artificial belief system

Proceedings of the International Joint Conference on Artificial Intelligence

Bedford, Mass. Mitre Corp. 1969

Coles L S

Syntax-directed interpretation of natural language
Carnegie Mellon University, doctoral dissertation 1967
and see also:
An on-line question-answering system with natural language and pictorial input
Proceedings of the 23rd National Conference of the Association for Computing Machinery 1968

Talking with a robot in English
Proceedings of the International Joint Conference on Artificial Intelligence
Bedford, Mass. Mitre Corp. 1969

Techniques for information retrieval using an inferential question-answering system with natural language input
Stanford Research Institute, Technical Note 74 1972

Fawcett R

Generating a sentence in a systemic functional grammar
University College, London mimeo 1973

Fillmore C J

The case for case
in: Universals in linguistic theory
Bach E & Harms R T (eds.)
New York 1968

Green B F, Wolf A K, Chomsky C, & Laughery K

BASEBALL: An automatic question-answerer

in: Computers and Thought

Feigenbaum E A & Feldman J (eds.)

New York McGraw-Hill 1963

Grinder J & Postal P

Missing Antecedents

Linguistic Inquiry 2 1971 269 - 312

Halliday M A K

Categories of the theory of grammar

Word 17 1961

Notes on transitivity and theme in English I

Journal of Linguistics 3 1967a

Notes on transitivity and theme in English II

Journal of Linguistics 3 1967b

Notes on transitivity and theme in English III

Journal of Linguistics 4 1968

Language structure and language function

in: New horizons in linguistics, ed. Lyons J

Harmondsworth Penguin Books 1970

An outlook on modern English

Oxford O.U.P. 1974

Homer

The Iliad

Oxford O.U.P. 1902

Huddleston R D

The sentence in written English: a syntactic study based on an analysis of scientific texts

London Cambridge University Press 1971

Hudson R A

English Complex Sentences

North Holland Linguistic Series, no. 4

London and Amsterdam 1971

Systemic generative grammar

University College, London mimeo 1972a

Systemic grammar simplified

University College, London mimeo 1972b

Isard S D

What would you have done if...?

Theoretical Linguistics (forthcoming)

Isard S D & Longuet-Higgins H C

Question-answering in English

University of Edinburgh, Theoretical Psychology Unit

mimeo 1970

Isard S D & Longuet-Higgins H C

Modal tic-tac-toe

in: Logic, Language, and Probability

edited by Bogdan R J & Niiniluoto I

Dordrecht Reidel 1973

Kaplan R M

Augmented transition networks as psychological models of sentence comprehension

Proceedings of the 2nd Joint International Joint Conference on Artificial Intelligence 1971

Katz J J

Generative semantics is Interpretive semantics

Linguistic Inquiry 2 1971

Kellogg C A

A natural language compiler for on-line data management

Proceedings of the Fall Joint Computer Conference 1968

New York Spartan 1968

Langacker R W

Chain of command

in: Modern Studies in English

edited by Reibel D A & Schane S A

Englewood Cliffs, N.J. 1969

Lindsay R K

A program for parsing sentences and making inferences about kinship relations

in: Symposium on simulation models

edited by Hoggatt A C & Balderston F E 1963a

Lindsay R K

Inferential Memory as the basis of machines which understand natural language

in: Computers and Thought

edited by Feigenbaum E A & Feldman J

New York McGraw-Hill 1963b

Longuet-Higgins H C

The algorithmic description of natural language

Proceedings of the Royal Society of London series B

182 255 - 276 1972

McCarthy J

Programs with common sense

in: Mechanisation of thought processes

National Physical Laboratory Symposium no. 10

London H.M.S.O. 1959

Power R

A computer model of conversation

University of Edinburgh doctoral dissertation, 1974

Quillian M R

Semantic Memory

in: Semantic Information Processing

edited by Minsky M

Cambridge, Mass. M.I.T. 1968

The teachable language comprehender

Communications of the Association for Computing

Machinery 12 1969

Raphael B

SIR: A computer program for semantic information retrieval

in: Semantic Information Processing

edited by Minsky M

Cambridge, Mass. M.I.T. 1968

Sachs J S

Recognition Memory for syntactic and semantic aspects of connected discourse

Perception and psychophysics 2 1967 437 - 442

Schank R C, Tesler L, and Weber S

SPINOZA II: Conceptual case-based natural language analysis

Stanford Artificial Intelligence Memo 109 1970

Schank R C

Finding the conceptual content and intention in an utterance in a natural language conversation

International Joint Conference on Artificial Intelligence 1971

Schultz J A & Bielby W T

An algorithm for the syntactic analysis in the R2 information system

Coordinated Science Laboratory, University of Illinois Report R-494 1970

Schwarcz R M, Burger J F, & Simmons R F

A deductive question-answerer for natural language inference

Communications of the Association for Computing Machinery 13 1970

Shapiro S C & Woodmansee G H

A net structure based relational question-answerer

Proceedings of the International Joint Conference on Artificial Intelligence

Bedford, Mass. Mitre Corp. 1969

Simmons R F

Storage and retrieval of aspects of meaning in directed graph structures

Communications of the Association for Computing Machinery

9 1966 211 - 214

1970 For Simmons (1970) see Schwarcz R M (1970)

Simmons R F, Burger J F, and Long R E

An approach towards answering English questions from text

Proceedings of the Fall Joint Computer Conference

New York Spartan 1966

Sinclair J McH

A course in spoken English: grammar

London Oxford University Press 1972

Tharp A L & Krulee G K

Using relational operators to structure long-term memory

Proceedings of the International Joint Conference on Artificial Intelligence

Bedford, Mass. Mitre Corp. 1969

Thompson F B et al.

DEACON: Direct English Access and Control

Proceedings of the Fall Joint Computer Conference 1966

New York Spartan 1966

Weizenbaum J

ELIZA

Communications of the Association for Computing Machinery

1966 9 36 - 45

Contextual Understanding by Computers

Communications of the Association for Computing Machinery

1967 10 474 - 480

Winograd Terry

Understanding Natural Language

Edinburgh Edinburgh University Press 1972

Woods W A

Procedural semantics for a question-answering machine

Proceedings of the Fall Joint Computer Conference 1968

New York Spartan 1968

Augmented transition networks for natural language analysis

Roport CS-1, Computation Laboratory, Harvard University

Cambridge, Mass. 1969

Woods W A, Kaplan R M, Nash-Webber Bonnie

The Lunar Sciences Natural Language Information System

Report 2378

Cambridge, Mass. Bolt, Beranek, & Newman Inc. 1972